Cadence Low Power Reference Flow

User Guide

for the

IBM-Chartered 90nm CMS9FLP Process

Version 1.4 (May 8th, 2006)

Table of Contents

List of Tables

List of Figures

1 INTRODUCTION

1.1 OVERVIEW

The IBM-Chartered 90nm Low Power Reference Flow was developed using the IBM 90nm CMS9FLP process information for the ARM® Artisan® Metro™ low-power libraries, and highlights the low-power features of the Cadence® Encounter™ digital integrated circuit (IC) platform. The design was implemented in the Cadence Encounter platform from RTL to GDSII, including RTL Synthesis, Silicon Virtual Prototyping, Physical Implementation and Signoff Chip Assembly.

The main components of the solution are:

IBM's 90nm process technology and the Cadence 90nm flow using the RTL-GDSII solution. The process information comes from IBM Micro-electronics, is ISO 9000 certified, and is committed to improving yield and reliability.

This low power reference flow solution has been validated as being compatible with IBM and Chartered for their 90nm joint enablement program with an optimized RTL to GDSII flow using Cadence tools that are optimized and tested using the IBM 90nm process technology and hence it reduces your design risk.

Another key benefit of the IBM Low Power Reference flow is the access to IBM Foundry's advanced design kits that are compatible with many well known design environments.

This tutorial goes through the entire RTL to GDSII flow, starting with synthesis using the RTL-Compiler technology, prototyping and floorplanning using First Encounter, flat implementation using SOC Encounter, and finally completes the flow with chip assembly and sign-off.

1.2 PURPOSE

This document provides the necessary information to run the Cadence Low Power Reference Flow for the IBM-Chartered 90nm CMOS process scripts. Also, it provides the information about the list of flows and the flow steps that were run in each flow.

1.3 TOOLS AND TOOL VERSIONS

Below is a listing of the specific tool versions required for proper execution of the low power reference flow:

Cadence® EDA Product	Version
Incisive™ Unified Simulator	IUS55
Encounter™ Conformal ASIC EC	CONFRML5.2 USR1
Encounter™ Test	ET 3.0.4 ISR
Encounter™ RTL Compiler	RC5.2 usr1
SoC Encounter GPS • First Encounter GPS • Nanoroute™ Ultra	SOC4.2 USR5
Celtic™ NDC (1)	v05.20-e113 (1)
VoltageStorm™ Dynamic Gate	ANLS 6.1 FCS
Fire & Ice QXC	EXT51 ISR200602070217

Table 1: Tools and Versions

Important Note (1): **This flow requires the use of a different version of Celtic™ NDC than that which comes bundled with the SoC Encounter GPS SOC4.2 USR5 software.** To insure that the proper version of Celtic™ NDC is used, the PATH environment variable must be modified to place the executable path for this specific Celtic™ NDC binary before the executable path for SoC Encounter GPS. This is taken care of through the use of the CELTIC_PATH environment variable in the reference flow setup scripts**. Prior to running this reference flow, the user must modify the value of the CELTIC_PATH in the following 3 setup files to reflect the location of this specific binary in the local software installation:**

- SETUP_IMPLEMENTATION.csh
- SETUP_SIGNOFF_SI.csh
- SETUP_SVP.csh

1.4 ARM METRO™ LIBRARY COMPONENTS

This 90nm low-power reference flow utilizes the following library components:

- 90nm CMS9FLP-HVT Process 1.2-Volt Metro™ v1.0 Standard Cell Library
- 90nm CMS9FLP-RVT Process 1.2-Volt Metro™ v1.0 Standard Cell Library
- 90nm CMS9FLP-LVT Process 1.2-Volt Metro™ v1.0 Standard Cell Library
- 90nm CMS9FLP-HVT Process 1.2-Volt SC-Metro™ v1.0 Multi-Voltage Kit
- 90nm CMS9FLP-RVT Process 1.2-Volt SC-Metro™ v1.0 Multi-Voltage Kit
- 90nm CMS9FLP-LVT Process 1.2-Volt SC-Metro™ v1.0 Multi-Voltage Kit
- CMS9FLP-HVT Process 1.2Volt Core, 2.5Volt Metro™ v1.0 General Purpose Staggered I/O Library
- Metro Single Port SRAM Generator, 90nm CMS9FLP HVT RVT Process: Version 2005Q4V1
- Metro Dual Port SRAM Generator, 90nm CMS9FLP HVT RVT Process: Version 2005Q4V2

1.5 INSTALLATION AND SETUP

To run the tutorial, install the tarkit and execute the following steps:

1) Check to make sure you have the following files with the correct size in KB:

```
1384    CommonPlatform_90LP_v1.0_FRONTEND.tar.gz
84236   CommonPlatform_90LP_v1.0_BACKEND.tar.gz
55944   CommonPlatform_90LP_v1.0_LIBS_celtic.tar.gz
40580   CommonPlatform_90LP_v1.0_LIBS_common.tar.gz
32324   CommonPlatform_90LP_v1.0_LIBS_qx.tar.gz
135632  CommonPlatform_90LP_v1.0_LIBS_sstorm.tar.gz
10808   CommonPlatform_90LP_v1.0_LIBS_tech.tar.gz
320336  CommonPlatform_90LP_v1.0_LIBS_vstorm.tar.gz

This file is only necessary if you want to have the results of the
backend steps:
2228248 CommonPlatform_90LP_v1.0_CHECKPOINT.tar.gz
```

2) Use the following commands to untar these files:

```
foreach i  (<PATH TO TARBALL>/CommonPlatform_90LP_*.tar.gz)
tar xvfz $i
end

OR

gtar xvfz <PATH>/CommonPlatform_90LP_v1.0_FRONTEND.tar.gz
gtar xvfz <PATH>/CommonPlatform_90LP_v1.0_BACKEND.tar.gz
gtar xvfz <PATH>/CommonPlatform_90LP_v1.0_LIBS_celtic.tar.gz
gtar xvfz <PATH>/CommonPlatform_90LP_v1.0_LIBS_common.tar.gz
gtar xvfz <PATH>/CommonPlatform_90LP_v1.0_LIBS_qx.tar.gz
gtar xvfz <PATH>/CommonPlatform_90LP_v1.0_LIBS_sstorm.tar.gz
gtar xvfz <PATH>/CommonPlatform_90LP_v1.0_LIBS_tech.tar.gz
gtar xvfz <PATH>/CommonPlatform_90LP_v1.0_LIBS_vstorm.tar.gz
gtar xvfz <PATH>/CommonPlatform_90LP_v1.0_CHECKPOINT.tar.gz
```

3) To begin change to the root directory:

```
cd CommonPlatform_90LP_v1.0
```

4) You should see 3 directories with the following size in KB:

```
4965764 BACKEND
21956   FRONTEND
2538608 LIBS

Without the checkpoint data it should be as follows:

246184  BACKEND
21956   FRONTEND
2538608 LIBS
```

2 DESIGN INFORMATION

2.1 OSPREY BLOCK DIAGRAM

The low power flow implements the Osprey design, using multi-Vt libraries and multiple voltage islands (see diagram). The dma and cipher blocks operate at 1.2V nominal (red), while the rest of the chip operates at the default 1.0V nominal. Both domains are non-switchable (i.e., "always on").

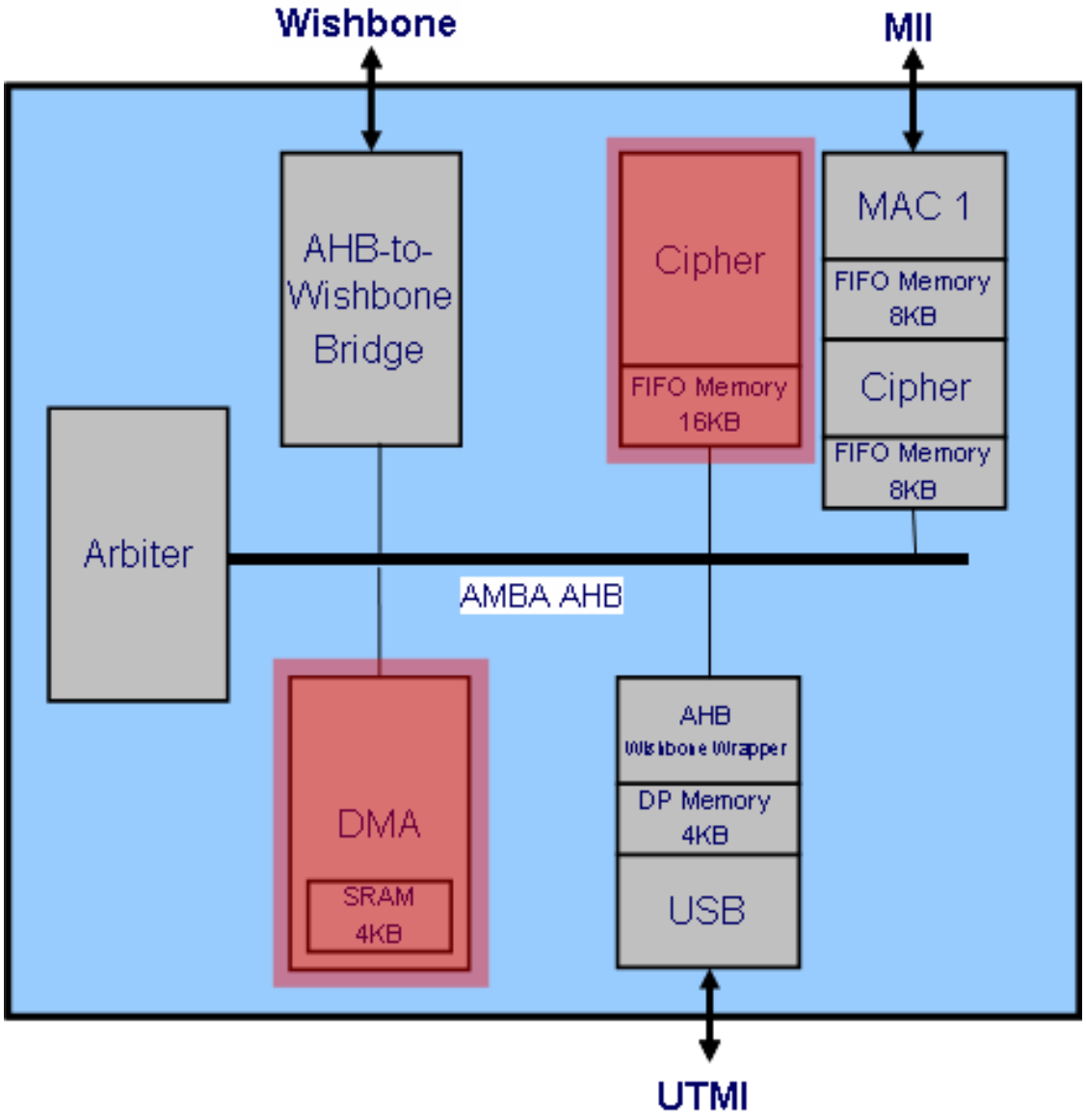


Figure 1: Osprey Block Diagram

2.2 OSPREY SYSTEM INTERFACES

The Osprey Design has the following system interfaces:

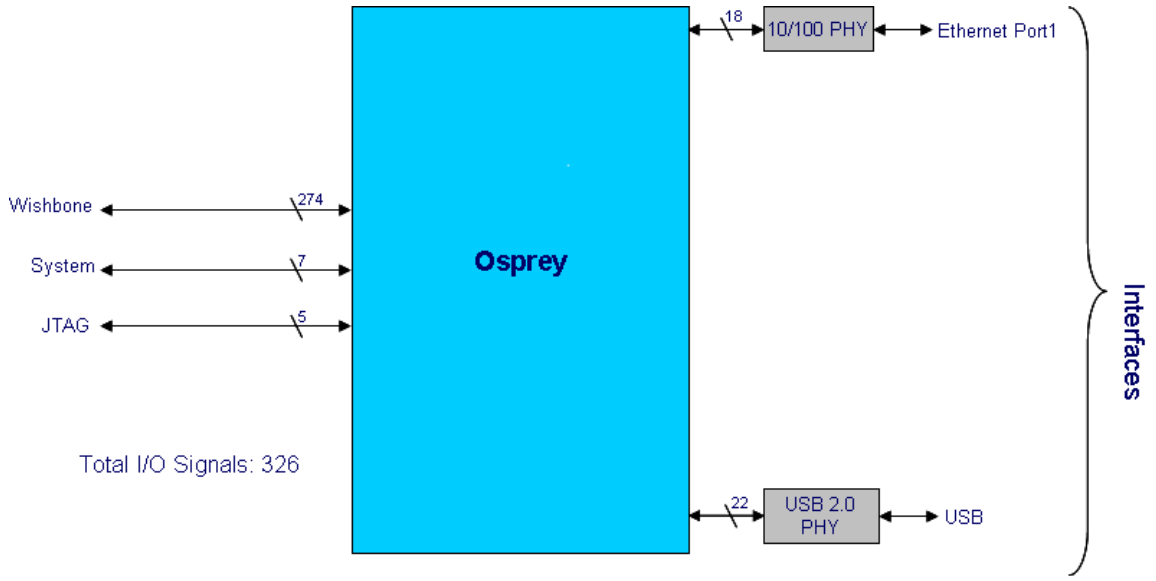


Figure 2: Osprey System Interfaces

3 FRONT END PROCESSING FLOW

3.1 PROJECT DIRECTORY SETUP

Go to the FRONTEND directory:

>% cd FRONTEND

You will see the following directories inside the FRONTEND directory:

Atpg	Scan test directory
bsv	Boundary scan test directory
func_test	Functional simulation directory
libs	Project specific library data
mbist	Memory BIST directory
src	Verilog source code directory
synth	RC synthesis directory
topshell	Topshell directory
lec	Formal verification directory

Next you will need to modify the variables in the SETUP.csh file in the FRONTEND directory as follows:

BASE	Root path to the FRONTEND dir.
LIBS	Path to library install (keep unchanged)
ET_PATH	Path to Encounter Test install root

These variables are needed to run all of the scripts that follow. You will then need to source this file before starting the front-end SVT flow by typing

>% source SETUP.csh.

3.2 BUILD TOPSHELL

The topshell build creates the I/O ring and inserts the JTAG TAP controller and boundary scan logic. This is done using 2 control files: **IOspeclist.optima** and **connection.optima**. The IOspeclist describes each I/O and the JTAG TAP controller. The connection file contains connections to the core and to the JTAG controller. Also in this directory is the **doBUILD_TOP_SHELL.csh** script that is used to build the topshell. The **libs.ovr** file is used to control which library cells are used when synthesizing the topshell logic.

In order to run the topshell script you must have your environment set up for Encounter Test (Version 3.0.4).

Run the following command from the topshell directory

>% doBUILD_TOP_SHELL.csh

This will create the file **osprey.v** in the **testresults/designsource** directory. This file will be combined with the core to build the complete netlist. The log file will be in the **testresults/logs** directory. It is called **log_build_top_shell**. For this design you should have only 1 Warning: (W) DFT165. This warning can be ignored.

3.3 SYNTHESIZE CORE

The next step is to synthesize the core logic. This will produce a gate level netlist without any test logic. The test logic will be added in a separate synthesis run. In order to demonstrate the low power features of RC we are using the following features:

- Clock gating
- Multiple VT libraries
- Multiple voltage islands
- Operand Isolation

The synthesis script is **synth/scripts/osprey_lps.tcl**. If you look at this script you will see that we have defined 2 library lists, a 1.0v library and a 1.2v library. Two of the blocks in the design, cipher and dma, are synthesized with the higher voltage libraries and the rest of the chip is synthesized with the lower voltage libraries.

This is done with the ***set_attribute library_domain*** command.

The clock gating is implemented with the command:
set_attr lp_insert_clock_gating true

The operand isolation is implemented with the command:
set_attr lp_insert_operand_isolation true

The multiple vt optimizations are implemented with the command:
set_attr lp_multi_vt_optimization_effort high

The **synth** directory contains several files and directories:

constraints	Directory contains timing constraints for the chip
log	Directory contains the log files from the synthesis run
netlists	Directory contains the chip netlists after synthesis
reports	Directory contains report files from synthesis
scripts	Directory contains synthesis scripts:
run_RC_lps	Initial synthesis script
run_RC_post	Synthesis script used to insert DFT
run_RC_toplevel	Toplevel synthesis script

This synthesis run will read in the RTL netlist from the **src** directory and produce a low power optimized gate level netlist. You must have the RC environment set up to run the synthesis. RC version 5.1 is used for this synthesis run.

Run the command from the **synth** directory. This command will call the tcl synthesis script **osprey_lps.tcl** in the **synth/scripts** directory.

>% run_RC_lps osprey

This will create the following files:

osprey_Core_elab.v	Elaborated Netlist
osprey_Core_generic.v	Design Mapped to Generic Netlist
osprey_Core_g2c.v	Synthesized design without MBIST or DFT or SCAN

And creates the following report files:

osprey_Core_g2c.power	Chip power broken down by blocks
osprey_Core_g2c.operiso	Operand Isolation report
osprey_Core_g2c.cgate	Clock Gating report
osprey_Core_g2c.area	Cell area report
osprey_Core_g2c.gates	Gates Used report
osprey_Core_g2c.timing	Timing report

3.4 ADD MBIST

This is done from the **mbist** directory. It uses the netlist **osprey_Core_g2c.v** as the input gate level netlist.

Run the command:

>% insert_embedded_test

This command uses the netlist, the BSDL file from the TOPLEVEL insertion, and the configuration file **cfg.osprey.txt** It will create a netlist with embedded MBIST in the **testresults/designsource** directory. The file name will be **osprey_Core_tem.v**. The log file is called **testresults/logs/log_insert_embedded_test**. If the run completes successfully the log file will have only warning messages related to the RETN pin on the RAMs, which can be ignored. After the core and toplevel netlists are combined we will verify the MBIST logic with a simulation run. The MBIST must be simulated at the chip level since it is controlled by the JTAG TAP controller which is in the toplevel netlist.

3.5 SYNTHESIZE TO ADD DFT

The next step is to run RC again on the netlist that includes MBIST to add the scan chain and test logic that is needed for ATPG testing. This is done after adding MBIST so the MBIST logic will be included in the scan chains. This step is run from the **synth** directory. This command will call the tcl synthesis script **osprey_lps_dft_postmbist.tcl** in the **synth/scripts** directory.

>% run_RC_post osprey

This will create the following files:

osprey_Core_elab_dft_postmbist.v	Elaborated Netlist
osprey_Core_g2c_dft_postmbist_nolevel.v	Synthesized Netlist with DFT, i.e., scan chains
synth/log directory	Contains the log file
synth/reports directory	Contains the synthesis reports

3.6 CREATE COMPLETE NETLIST

This step merges the toplevel netlist that contains the I/O cells and the JTAG logic with the core netlist to create the complete chip netlist. This is run from the **synth/netlists** directory.

>% create_full_netlist

This file is now the complete netlist with the core, I/O cells and JTAG logic.

3.7 TOPLEVEL SYNTHESIS RUN

This RC synthesis run is done to add level shifters for the multiple voltage domains and to remove any assign statements from the netlist. It is also used to write out a MSV control file for FE. This is needed so that FE will have the multiple voltage island information. This is run from the **synth** directory

>% run_RC_toplevel osprey

This will create the following files:

osprey_elab_toplevel.v	Elaborated Netlist
osprey_g2c_toplevel_NOlevelshifters.v	Final Netlist without Level Shifters
osprey_g2c_toplevel_levelshifters.v	Final Netlist with Level Shifters

Other files created:

osprey_g2c_toplevel.power_dft	Power Report
osprey_g2c_toplevel.levelshift_dft	Level Shifter Report
osprey_g2c_toplevel.area_dft	Area Report
osprey_g2c_toplevel.gates_dft	Gate Usage Report

FE_IMPLEMENT_TOPLEVEL2 Directory containing files for Multiple Voltage domains. The files in this directory are used by FE to identify and setup the multiple voltage domains.

At this point the netlist **osprey_g2c_toplevel_levelshifters.v** is the complete netlist that will be taken in to FE for place and route. The remainder of the flow involves the steps required to verify the MBIST logic, ATPG (scan) logic, and BSV (boundary scan) logic. We also need to run CONFORMAL LEC to make sure that the final netlist is logically equivalent to the original RTL source. We will also run functional simulations on the final netlist.

3.7.1 Creating Toplevel ScanDef file

In order to create a toplevel ScanDef file you must edit the netlist **osprey_g2c_toplevel_levelshifters.v** This file is in the **synth/netlists** directory. Search for any 'inout' pins in any module with the name mbist*. Change these pins from 'inout' to 'output'. Then write out the netlist with the name **osprey_g2c_toplevel_levelshifters_mfix.v**.

The next step is to run a final RC run to generate ScanDef file. This script is run from the synth directory.

>% run_RC_toplevel_scandef osprey

This will create the ScanDef file in the netlists directory called **`toplevel_scandef.def`**.

3.8 MBIST SIMULATIONS

The MBIST simulations verify the operation of the RAM Memory BIST. MBIST is controlled through the JTAG TAP controller. In this step we will generate a Verilog test bench using Encounter Test and then use NCsim to simulate the MBIST logic. The MBIST simulations are run on the toplevel netlist with the I/O and JTAG logic inserted.

First we need to run 3 scripts to generate the Verilog testbench for MBIST.

Change to the **`mbist/testresults/scripts`** directory

Copy the scripts from the mbist/scripts directory to this directory. This step is needed to fix some problems with the script generation in this version of Encounter Test.

>% cp ../../scripts/run.osprey.1149_tmb .

>% cp ../../scripts/run.osprey.TDR_special_tmb .

>% cp ../../scripts/run_sim.osprey.definers .

Run the 3 scripts to create the verilog test bench:

Start Encounter test in command line mode:

>% et -c -f

Run the first script:

>% run.osprey.1149_tmb | tee 1149_tmb.log

The log file should only contain 1 (E) error message:

TJB207(E) No constant PIN_MAP_STRING statements are defined.

This message can be ignored.

Run the second script:

>% run.osprey.TDR_special_tmb | tee TDR_special_tmb.log

The log file should only contain 1 (E) error message:

TJB207(E) No constant PIN_MAP_STRING statements are defined.

This message can be ignored.

Run the third script:

>% run_sim.osprey.Definers | tee Definers.log

The log file should only contain 1 (E) error message:

TJB207(E) No constant PIN_MAP_STRING statements are defined.

This message can be ignored.

The verilog testbench files will be in the **`mbist/testresults/embedded_macros/dv/verilog`** directory

Change to the **`mbist/testresults/embedded_macros/dv/verilog`** directory

Copy the run scripts to the **`testbench`** directory:

>% cp ../../../../scripts/run* .

Start up the Ncsim simulation environment.

There is a **`runall`** command that will run all 8 simulations.

>% runall

The testbenches are self checking. If you look at the log file for each simulation at the end there will be a list of the Miscompares. The Miscompare number should be 0 for all of the bypass simulations. There will be errors in the non bypass simulations. These are because the design needs to be simulated with the backannotated SDF to eliminate timing issues. The log files have a .log extension and are in the current directory.

3.9 ATPG

This step is used to create the ATPG scan vectors. This is run from the **atpg** directory. There are 12 scripts that are run to create and verify the scan vectors for this design.

In this directory there is a **runall** script that will run all 12 scripts. The Encounter Test environment must be set up to run these scripts.

doBUIILD_LOGIC_MODEL.csh	Reads the netlist and builds the logic model
doBUILD_TEST_MODE.csh	Creates the Scan test mode
doTSV.csh	Runs the Test Structure Verification checks
doTS_REPORT.csh	Generates a Testablility report
doBUILD_FAULT_MODEL.csh	Creates the fault model
doCREATE_ATPG_TESTS.csh	Creates the Test vectors
doCOMPACT_VECTORS.csh	Compresses and orders the Test vectors
doWRITE_VLOG_VECTORS.csh	Writes out the vectors in Verilog format
doWRITE_WGL_VECTORS.csh	Writes out the vectors in WGL format
doREPORT_VECTORS.csh	Generates Vector report file
doREPORT_FAULTS.csh	Generates Fault report file
doREPORT_VECTOR_STATISTICS.csh	Generates statistics file

To run these scripts, either execute the **runall** script:

>% runall

Or run each script individually:

>% doBUILD_LOGIC_MODEL.csh
>% doBUILD_TEST_MODE.csh do
>% TSV.csh
>% doTS_REPORT.csh
>% doBUILD_FAULT_MODEL.csh
>% doCREATE_ATPG_TESTS.csh
>% doCOMPACT_VECTORS.csh
>% doWRITE_VLOG_VECTORS.csh
>% doWRITE_WGL_VECTORS.csh
>% doREPORT_VECTORS.csh
>% doREPORT_FAULTS.csh
>% doREPORT_VECTOR_STATISTICS.csh

- The log files are in the **testresults/logs** directory.
- The Verilog vectors are in the **testresults/verilog** directory.
- The WGL vectors are in the **testresults/wgl** directory.

3.10 BSV VERIFICATION

The Boundary Scan Verification simulations are performed in the **bsv** directory. There are 4 scripts that are run to build the BSV testmode and generate the Verilog testbench.

>% cd to the `bsv` directory.

There is a command called **`run_scripts`** that will run all 4 scripts:

>% run_scripts

The first script is called **`MODEL.csh`**. This will build the Encounter Test model for the design. The log file will be put in the **`testresults/logs/log_build_model`** file. You will see the following non-fatal error messages which can be ignored:

ERROR Messages…
2 ERROR – Cell named RAM_1024_32 appears to be a RAM or ROM but…

The second script is called **`TEST_MODE.csh`**. It is used to build the 1149 test mode. The log file is **`testresults/logs/log_build_testmode_1149`**.

The third script is called **`BOUNDARY_TSV_11491.csh`**. This step runs the boundary scan verification. The log file is in **`testresults/logs/log_verify_11491_boundary_1149_11491_expt`**. In the log file you should search for the section titled "BSV Simulation Results Summary." You should not see any miscompares in this section. You will see an error saying that Boundary Scan Verification failed. This is because we are not testing the MBIST instructions in this run. The MBIST instructions are verified during the MBIST simulations.

The last script is called **`WRITE_JTAG_VLOG.csh`**. This step generates the Verilog testbench to be used for the simulation of the Boundary Scan.

The log file for this step is **`testresults/logs/log_write_vectors_1149_11491expt_*`** (where * is a date code).

The next step is to run the verilog simulations. The verilog testbench is in the **`testresults/verilog`** directory. You will need to copy the 6 **`run_verilog*.tcl`** scripts to the **`testresults/verilog`** directory. These six scripts will run ncverilog for the 6 testbenches that were generated.

>% run_verilog1.tcl
>% run_verilog2.tcl
…
>% run_verilog6.tcl

The log files will be called **`verilog1.log … verilog6.log`**.

3.11 CONFORMAL LEC CHECKS

Formal verification is done in 2 steps. The first step is to verify the RTL netlist against the first netlist from synthesis. This netlist will not have any DFT added. The second step is to verify the first netlist from synthesis against the final netlist that is delivered to place and route.

The Conformal checks take place in the **`lec`** directory.

There are 3 subdirectories under **lec**:

etc	Contains the dofiles to run LEC
log	Contains the log files from the LEC runs
rpt	Contains the report files from the LEC runs

The LEC checks are run from the **lec** directory.

>% lec –dofile etc/rtl2predft.do
>% lec –dofile etc/gate2gate.do

The files **rtl2predft.do** and **gate2gate.do** are the control files that set up the LEC runs. These files read in the netlists, constrain any pins that must be held to a particular value, and perform the compare. The results will be in the **log** files and in the **rpt** directories.

3.12 FUNCTIONAL SIMULATIONS

The functional simulations are Verilog simulations. The **func_test** directory contains the functional simulations. The simulations are run from the **sim** directory. The simulations are run on the gate level netlist. There is a script (**runall_gate**) that will run the 5 runs for the netlist.

In the **func_test/sim** directory there is a file called **makefile** that is used to run the simulations. There is a file in **func_test/sim/etc** called **osprey_gate.f**. This file contains the list of files needed for each simulation. In this file you will see the design files, library files and testbench files. You will need to make sure that the library files are correct for your runs.

To run the simulations, make sure your NCVerilog environment is enabled.

>% runall_gate

The testbenches are self checking. Any errors will show up in the ncsim log files in the **func_test/sim/log** directory.

4 IMPORTANT TECHNOLOGY AND LIBRARY CONSIDERATIONS/OPTIMIZATIONS

4.1 TECHNOLOGY CONSIDERATIONS

It is important to review the library documents and appnotes delivered by ARM, as well as the physical rule documentation from IBM.

- Level shifters for this library are always placed in the receiving power domain. In our case LVLLH* (low to high) shifters are placed in the hi_power domain, while LVLHL (high to low) are placed in the lo_power domain. Also note that the LVLLH* are 3x the standard cell height (please check the LEF modification part of this document) and have access to both power rail. In contrast the LVLHL* level shifters are single height and access only the low power domain power rail.
- Endcap cells need to be inserted for this library/process in order to close the well continuity at the end of the placement rows. Endcaps always have to be the first element to be inserted. All endcaps need to be paired vertically, which means that an even number of rows is required in the design and in the power domains. Any cut in these rows also need to be the size of an even number of rows. We provide two procedures in **`checkEndcap.tcl`** that can help satisfy these requirements. These requirements are difficult to meet. We also recommend using a user grid with a vertical pitch of 2x the row height. You might consider turning on some snapping features, but keep in mind that the size of blockages, including instance halo blockages have to be taken into consideration. The procedures provided will automatically adjust the blockage and halo sizes, as well as the power domain size and gap.
- This library is tapless, which means that the standard cells do not include power and ground connections to the wells and substrate. As a result it is the user's responsibility to add well tap cells to satisfy the IBM physical design rules. Since well taps have to follow strict spacing rules they have to be inserted directly after the endcaps (note that some endcaps include well taps).
- The power routing of the second rail (VDDI) of the triple height shifters (LVLLH*) pauses additional placement challenges. To avoid shorts, no standard cells should be placed between the level shifters and the power domain boundary. This requires special setPlaceMode settings in Encounter.
- Since well taps are placed before the level shifters some taps will be placed between the power domain edge and the level shifters. The VDDI power rail pin is located on the second standard cell row of the triple standard cell height shifters. Hence we have an additional constraint on the well tap placement to never place a well tap on the middle standard cell row of these shifters. This is done by carefully choosing the vertical spacing and offset of the well tap insertion command.
- Finally, since endcap cells are also inserted before the level shifters, and need to be placed at the end of every row without exception, we need to use a special endcap cell in the power domain where the triple height shifters are located. The standard endcap cells would create a short. The special version of the endcap cell is tapless and has no pin or blockage that would create a short when connecting the VDDI power rail of the triple height shifters.

4.2 REQUIRED LEF FILES OPTIMIZATIONS

4.2.1 Shifter Site

The original LEF files do not contain a dedicated SITE for the triple height level shifters. Without this site these level shifters can be placed on any standard cell row and could sometimes stagger and cause shorts during the power rail connection. To remedy this problem the following SITE definition should be added (we added them to the level shifter LEF for each VT).

```
SITE SHIFTERSITE

   SYMMETRY Y  ;

   CLASS CORE  ;

   SIZE 0.280 BY 5.88 ;

END SHIFTERSITE
```

In this flow we have inserted the level shifters in the netlist during synthesis. First Encounter will detect them and flood the core with rows for the SHIFTERSITE type.

If there were no level shifters in the netlist, then we would have had to submit the following command, where llx, lly, urx, ury are the coordinates of the power domain where the triple height shifters should be placed (there is no harm in defining rows over the entire core).

createRow -site SHIFTERSITE -area <llx lly urx ury>

4.2.2 IO LEF

The LEF definitions of the PAVDD_S/T and PAVSS_S/T pads should be modified such that the AVDD and AVSS pins are of type USE POWER and USE GROUND respectively and not USE SIGNAL.

Without these modifications Sroute will not be able to connect to these power pins.

4.2.3 generateLef command and multicut vias

Use the ***generateLef*** command to add more multicut vias to the technology LEF. The more multicut vias defined the higher the density of multicut vias inserted during routing, hence the higher the yield.

Even more multicut vias can be added by manually defining them. Because of the higher routing utilization on metal1 due to pins and blockages in the standard cells it is typically harder to get a high count of multicut vias on the lowest via, yet this layer contains a very high density of via cuts compared to other layers thus directly impacting the overall multicut via count.

Note however that the number of multicut vias defined directly impacts the routing runtime. Because of the above metal1 density considerations we recommend adding a reasonable number of lower level multicut via definitions, but limit the number for the higher levels.

There is also a limit of 30 vias per cut layer that can be defined in the technology LEF.

4.2.4 VIA single layer TURN rules

These are not required anymore and should be deleted from the technology LEF to prevent unnecessary warnings in the log file.

4.2.5 PARALLELOVERLAP Statement

In order to satisfy physical design rules 553 and 553b the cut layers definition need to use a newer LEF syntax. This syntax will be available in LEF version 5.7; however this version hasn't been released yet. There's however a mechanism to pass the new syntax as property definitions.

Please follow these steps:

- change the VERSION statement in the technology LEF from 5.5 to 5.6
- at the top of the technology LEF insert:

```
PROPERTYDEFINITIONS

    LAYER LEF57_SPACING STRING ;

END PROPERTYDEFINITIONS
```

- Then for each cut layer add the following line, with the appropriate value for the given cut layer. For example for layer V1:

```
PROPERTY LEF57_SPACING "SPACING 0.180 PARALLELOVERLAP ;" ;
```

4.2.6 Via farms

The original technology LEF file does not support via farms rules. These rules state that large via arrays should be split in multiple smaller via arrays. The corresponding rule numbers in the IBM physical rule documentation are 551aR, 551bR, 551cR. Note that these rules are part of the recommended rules of the physical verification deck, so unless you turn on the recommended rule switch in the verification deck they will not be verified by the physical verification software.

At this time First Encounter only supports one of these rules at a time, not all three. This restriction will not cause a drc violation, but it will slightly limit the number of cuts per array. For greater flexibility Cadence and IBM recommend using the 4xN rule. In future releases of First Encounter all three rules will be available concurrently.

Since version 5.7 of the LEF permitting the definition of this rule has not been released yet, a similar mechanism to the PARALLELOVERLAP syntax needs to be used to pass this information. To achieve this please follow these steps:

- in the PROPERTYDEFINITIONS section defined earlier, add the following line LAYER LEF57_ARRAYSPACING STRING ;
- for each cut layer up to V5 add this line:

PROPERTY LEF57_ARRAYSPACING "ARRAYSPACING 0.28 ARRAYCUTS 4 ;" ;

4.2.7 SAMENET Statements

All the SAMENET statements should be removed from the technology LEF. Some of them could limit routing efficiency, while not providing any benefit, some others are default, and some others are an incomplete attempt at achieving what's now performed by the PARALLELOVERLAP statement mentioned above.

4.3 ENCOUNTER SETUP FILES

4.3.1 Footprint file

It is necessary to dump a footprint file with the following command:
"***reportFootprint –outfile osprey.footprint***"

This is necessary because the .lib files have the delay cells split between 4 footprints: dly1, dly2, dly3, and dly4. Instead we need to make a single footprint for all delay cells. In the footprint file replace dly1, dly2, dly3, and dly4 by a generic dly.

The footprint file can be found in **`SOURCE/osprey.footprint`**

4.3.2 Capacitance table and capacitance multipliers

It is assumed that the user is familiar with the generation of capacitance tables to be used with First Encounter (using ***generateCapTbl*** from the Unix prompt).

Capacitance multipliers also need to be determined using "***generateRCFactor –all –useOstrich***" in First Encounter. It is important to note that these factors are design and floorplan-dependant, so they should be regenerated accordingly.

4.3.3 MSMV setup file and level shifter file

The MSMV file contains information regarding the power domains. The MSMV setup file generated during the front-end flow should be considered as a template. Not all the necessary information is available during the synthesis run. This information needs to be added to the setup file before it can be used for back-end implementation. Please compare the **`osprey_RC_FE_MSV.setup`** file provided in the **`SOURCE`** directory with the one that was generated during synthesis. Among other changes the best case libraries need to be listed and the proper power and ground names need to be given.

Some physical information like gap spacing for the power domain can be added.

The level shifter file defines the direction of the level shifters. They also let the tool know in which power domain they should be placed.

The level shifter file can be found in **`SOURCE/osprey.vsf`**

4.3.4 Miscellaneous files

There are other files involved in this flow such as the clock tree definition file (cts or ctstch file), scan DEF, or IO file. There is no low power or library specific information in these files. It is assumed that the user is familiar with them.

5 BACKEND FLOW OVERVIEW

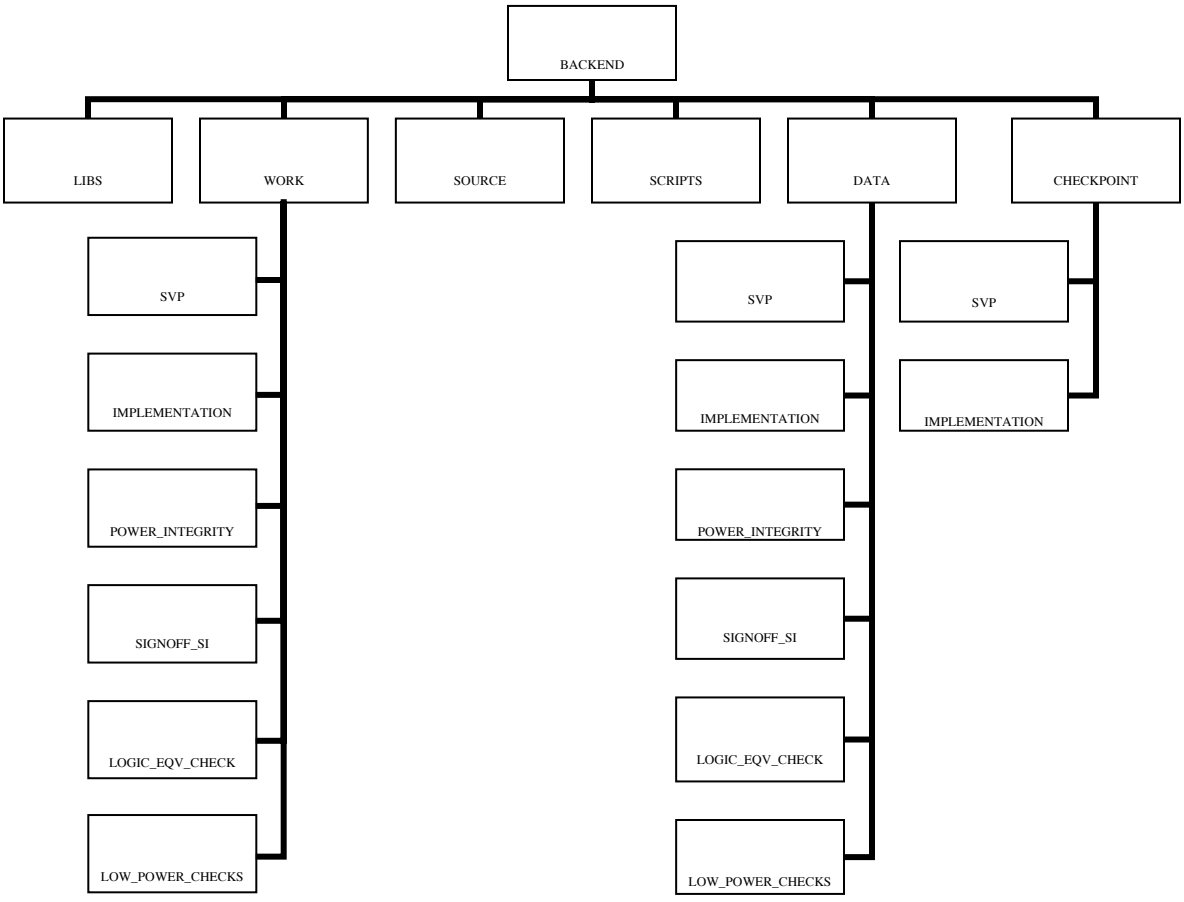
The backend flow is split between the virtual prototyping flow (SVP), the implementation flow, and the signoff flow. The signoff flow includes logical equivalency check, low power checks, static and dynamic power-rail analysis, and signoff signal integrity analysis.

All three sub flows share the same directory structure which is described below.

Please note that it is very important that the user gets familiar with the previous section (**4: Important Technology and Library Considerations/Optimizations**) regarding technology considerations and LEF optimizations.

It is assumed that the user is familiar with the concepts of backend implementation and has some knowledge of the Encounter software. This flow will focus on the specific features and techniques required to implement a multi voltage multi supply design (MSMV) using the ARM Metro libraries for the IBM/Chartered CMOS9FLP process.

5.1 BACKEND DIRECTORY STRUCTURE



Note that the **CHECKPOINT** directory contains the results of some steps in the flow. This way the user can load the results of steps that would have otherwise taken a long time to complete. The steps that have the longest runtimes are the ones in the implementation flow. The **WORK** directory is the location from which to launch the run scripts. The user should go into the **WORK** subdirectory matching the current task. All log and run files will be collected here. The **DATA** directory will receive the results of each step in the flow.

5.2 THE DBGLINKSIBLINGSTOGETHER VARIABLE

For the version of SoC Encounter GPS used in this reference flow (see section **1.3: Tools and Tool Versions**), it is important to set the **dbgLinkSiblingsTogether** variable to **1** prior to reading the configuration file. This variable setting is not saved with a database, so it should also be set before restoring a database. This is taken care of in the reference flow scripts, but users must be cognizant of it if they deviate from the flow scripts, or when running on their own data with this version of the software.

This variable setting is necessary to allow cells of a given Vth to be swapped to a cell of another Vth. Without this variable setting, a cell is allowed to be upsized to a cell with a higher drive strength, but the Vth of the cell would remain the same. In that case, the design will likely never meet timing, and local congestion would be severely impacted since the tool would automatically choose cells of higher drive strength, which are also physically larger.

In a future release, this variable is expected to be set as the default.

6 SILICON VIRTUAL PROTOTYPING (SVP)

The intent of the virtual prototyping flow is to generate the best floorplan possible which would then be passed on to the implementation flow. Virtual prototyping is an iterative process involving many trade-offs and choices. Here we're going to directly show how to come up with a possible floorplan, thus bypassing most iterations. However the steps involved in the setup and analysis of any given iteration are demonstrated. Some of the interactive steps will appear in their scripted version.

It is highly recommended that, at least for the first pass through the implementation flow, the user load the saved databases provided with this flow so that the results will be deterministic and can be compared with our results. After that, the user should feel free to explore the design and come up with alternative solutions.

The overall flow we are demonstrating here is flat; however the virtual prototyping stage for a hierarchical flow would be very similar. It is a unique advantage of First Encounter to be able to deal with multi voltage islands in a flat manner, because even a hierarchical implementation approach would require a flat virtual prototype.

6.1 RUNNING THE FLOW

The virtual prototyping flow goes through the following steps:

- zero load STA
- setup of all the low power libraries, global connections, and level shifter definitions
- floorplan generation
- placement of all cells likely to affect placement analysis (endcaps, well taps, jtag)
- low effort placement
- low effort IPO
- clock tree synthesis
- power analysis

Going through these steps allows the user to assess the quality of a floorplan, constraints.

To run the flow:

(**Note**: prior to running the flow, review section **5.2: The dbgLinkSiblingsTogether Variable**.)

>% cd WORK/SVP

>% source ../../SCRIPTS/SETUP_SVP.csh

>% ../../SCRIPTS/svp.csh

The ***svp.csh*** file will first run zero load timing analysis in a separate encounter session, uniquify the netlist, and then call First Encounter with the **`svp.tcl`** script.

Please review the **`svp.tcl`** script and the sequence in which it calls the different step scripts.

Some of these step scripts are the same that will be used during the implementation flow. Others have only been modified to lower the effort level since virtual prototyping is not about design closure but design investigation.

Some of these scripts should be carefully reviewed as they are essential for meeting all rules for this process and libraries.

Note that during floorplanning we recommend defining a user grid with a Y value of 2x the standard cell row height. This will simplify implementation and verification of the row pairing requirement.

The user should also turn on the visibility of screen densities as we make use of them in the proposed floorplan.

6.2 IMPORTANT SCRIPTS

6.2.1 init.tcl

The **`init.tcl`** script is called within the step scripts and is meant to be run by the first step script in a given session.

It initializes most mode settings such as ***setPlaceMode***, ***setOptMode***, etc...

It also contains the ***setOpCond*** command that sets the operating conditions for the design on a per power domain basis. The ***setOpCond*** data is NOT saved in the database, contrary to most mode settings.

6.2.2 floorplan.tcl

The **`floorplan.tcl`** script is meant to create the floorplan that will be used throughout the flow.

Please review it carefully as it includes some important comments. In order to understand why some commands are issued in this script it is important to read section **4: Important Technology and Library Considerations/Optimizations** regarding important technology considerations.

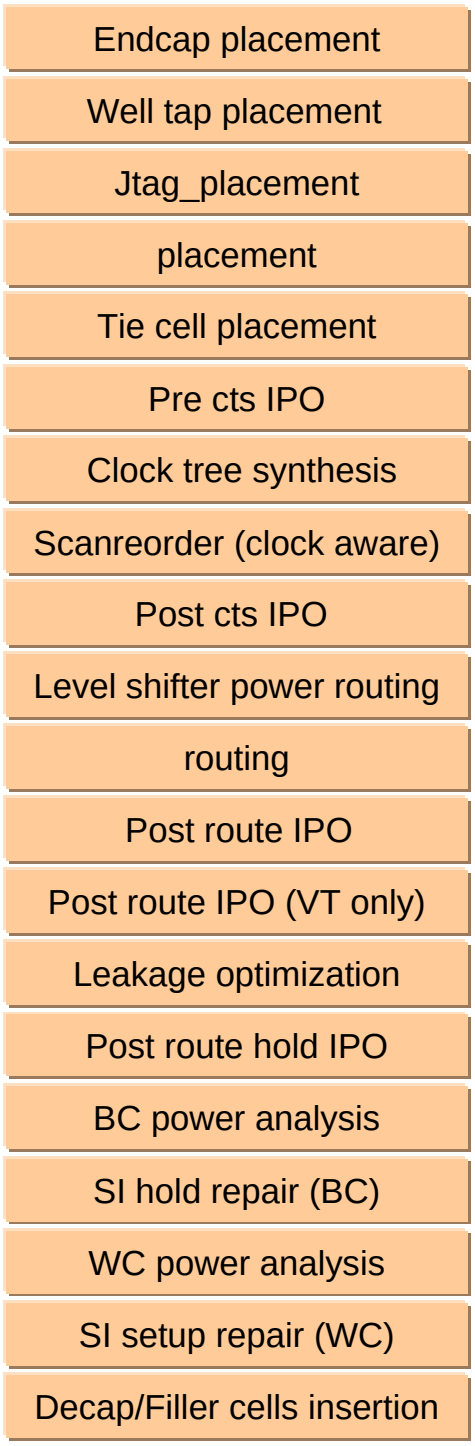
6.2.3 checkEndCap.tcl

This **`checkEndCap.tcl`** file defines two procedures that help with the implementation and verification of a floorplan that meets the row pairing requirement for this technology. Please refer to section **4: Important Technology and Library Considerations/Optimizations** for explanation about the row pairing requirement.

- ***checkBlockageEvenRow*** will make some minor modifications to the floorplan so that it meets the row pairing requirement. Placement blockages, instance halo blockages, power domain and their associated gap spacing are floorplan objects or values that could be modified.
- ***checkEndcapCell*** <endcapCellName> will verify that each endcap cell is paired correctly. EndCap cells need to be inserted prior to calling this procedure. If different endcap cells are used in different power domains then the procedure needs to be called for each endcap cell.

7 IMPLEMENTATION

7.1 FLOW DIAGRAM



7.2 RUNNING THE FLOW

Most of the steps in the flow are listed in the diagram above. At the end of the implementation flow all the data files necessary for the signoff flow will be saved. These include signoff extraction spef files for best RC and worst RC corners, logical and physical verilog, MSMV files for celtic, DEF file for voltage storm.

To run the flow:

>% cd WORK/IMPLEMENTATION

>% source ../../SCRIPTS/SETUP_IMPLEMENTATION.csh

>% ../../SCRIPTS/implementation.csh

The implementation.csh file will first run zero load timing analysis in a separate encounter session, uniquify the netlist, and then call First Encounter with the **`implementation.tcl`** script.

Please review the **`implementation.tcl`** script and the sequence in which it calls the different step scripts.

Some of these scripts should be carefully reviewed as they are essential for meeting all rules for this process and libraries.

Important Note (please refer to section **1.3: Tools and Tool Versions**): A different version of Celtic than the one packaged with SOC42USR5 is required for this flow to work properly. In order to have First Encounter call the correct version of Celtic during the SI repair loops the user needs to update the CELTIC_PATH environment variable in the **`SETUP_IMPLEMENTATION.csh`** script before starting the flow.

(**Note**: prior to running the flow, also review section **5.2: The dbgLinkSiblingsTogether Variable**.)

Since the entire First Encounter implementation flow takes around two days to complete on a fast machine, we have provided a number of checkpoint databases in the **`CHECKPOINT/IMPLEMENTATION`** directory. Databases for all the steps that require significant runtime are provided. A number of scripts are available in the **`SCRIPT`** directory that will restart the flow after a given checkpoint. These scripts are named **`after__<step>__checkpoint.tcl`**. To run any of the checkpoint scripts please do the following:

>% cd WORK/IMPLEMENTATION

>% source ../../SCRIPTS/SETUP_IMPLEMENTATION.csh

>% encounter ../../SCRIPTS/after__<step>__checkpoint.tcl

7.2.1 Highlights of Critical Concepts and Commands

Some of these scripts should be carefully reviewed as they are essential for meeting all rules for this process and libraries. You will notice calls to the ***verifyPowerDomain*** command at the end of any step which might change cell placement or cell types. This command will report any problems related to power domains.

7.2.2 Endcap, Well Taps, and Jtag Insertion

Endcap cells should always be inserted first. As discussed in section 4**: Important Technology and Library Considerations/Optimizations**, the endcap cells always need to be inserted in pairs. This is a floorplan requirement which has to be addressed during the silicon virtual prototyping flow. The **`add_endcap.tcl`** script will add the endcap cells on a per power domain basis. Note that we use a different type of endcap cell for hi_power domain.

ENDCAPMHT, which is used in the default lo_power domain, is the standard endcap cell and includes well taps.

ENDCAPBBMHT which is used in the hi_power domain doesn't include well taps. It is necessary to use this type of endcap cell in the domain that will contain the triple height level shifters. This is due to the second power rail of the level shifter (the VDDI pin which in our case connects to VDD_lo_power) that needs to cross the power domain boundary and connect to the ring surrounding the power domain. Only the ENDCAPBBMHT version can be routed over by the VDD_lo_power rail without creating a short.

The well taps insertion should always be inserted right after the endcap cells as there is a strict placement rule between well taps and active diffusion. As mentioned in section 4**: Optimizations, we** need to be cognizant of the level shifter placement when defining the vertical well tap spacing. If this is not done properly a power short could occur between the well taps and the triple height shifter VDDI pin (VDD_lo_power) connection. In order to avoid shorts the well taps should never fall on the center row of the triple height shifters.

Jtag placement should logically follow endcap and well tap insertion. Note that in the **`place_jtag.tcl`** script we call a procedure userUnfixJtagCells. This procedure will change the fixed placement status of the jtag instances thereby allowing IPO on them. For some designs this procedure might not be useful.

7.2.3 Placement and Scan Reordering

In the **`init.tcl`** script which is read entirely once per session we defined the following placement options.

setPlaceMode -clkGateAware -maxDensity 0.8 -highEffort -timingDriven -blockedShifterRows -noDividedShifterRows -maxShifterDepth 50 -ignoreScan –screenFix

These options will affect placement, but also all refine place calls made from many commands throughout the flow.

Description of some of these options:

- ***clkGateAware*** requires that the clock definition file (ctstch or cts file) be read prior to placement. It will place clock gating components such that they are in the center of gravity of the clock pins they drive.
- ***maxShifterDepth 50*** will prevent level shifters from being placed more than 50um away from the edge of the power domain. This simplifies routing of the power rails to the triple height shifters, and as a result helps routing congestion.
- ***blockedShifterRows*** will prevent standard cells from being placed in the space between two level shifters. This will prevent power routing shorts when connecting the VDDI pin of the triple height level shifters. Using this library we are bound to have some gaps in between some shifters because well tap cells are placed before the level shifters and cannot be moved.
- ***noDividedShifterRows*** will give better quality of results. It allows shifters to be placed anywhere on the top and bottom rows of the hi_power domain (or the rows on the other side of the power domain gap in the lo_power domain) instead of forcing them to the left and right corners. Note that this option combined with the –blockShifterRows option above can lead to a significant number of standard cells being pushed out of the area between level shifters. For this reason it is recommended to place a 0% density screen (or soft blockage) on the bottom rows of the power domain (in this case) where shifters are expected.
- ***screenFix*** is a temporary option which will be turned on by default in a later release of First Encounter. It was added to take advantage of a placement bug fix when using density screens in conjunction with power domains.

Placement is performed by the placeDesign option. By reading the scanDef definition file we turn on scan reordering. Please note that ***setScanReorderMode –msmv*** is required for MSMV designs in Encounter 4.2, but will not be necessary in Encounter 5.2

7.2.4 Timing Optimization

Logic optimization for timing and area is performed at different stages of the design: after placement, after CTS, and after routing. These steps are standard with any implementation flow. Here are a few important points specific to low power designs.

Some of the ***setPlaceMode*** options defined in the **`init.tcl`** script and explained in the placement section (**7.2.3Placement and Scan Reordering**) also affect the refine placement stages of optimization. For example, the options related to level shifters are important.

After placement, but before running the first optimization ***setPlaceMode –fixedShifter*** needs to be issued. This will ensure that level shifters will be in fixed status after the first optimization. This is the best place to fix the level shifters. By fixing those after clock tree synthesis there would be a chance that level shifters are refinePlaced past fixed clock instances (clock buffers, leaf registers, clock gates).

The ***–optimizeNetsAcrossDiffVoltPDs*** option of ***setOptMode***, which is default in recent versions of Encounter, enables optimization of paths that cross power domains.

Please check the **`pre_cts_opt.tcl, post_cts_opt.tcl`**, and **`post_route_opt.tcl`** scripts. For more information about the optDesign command please refer to the First Encounter documentation.

7.2.5 Clock Tree Synthesis

A lot of the clock tree power efficiency for the design depends on the front end flow with options regarding clock gating insertion. However there are a number of things that should be considered in the backend implementation.

Below is the list of options that will drive the clock tree synthesis step (**`cts.tcl`**). Make sure to review the **`SOURCE/osprey.cts`** clock tree definition file as well.

The options passed to the clock tree synthesis engine are as follow:

setCTSMode -ocvAwareCluster -usePowerInCTS -bottomPreferredLayer 3 -topPreferredLayer 6 -preferredExtraSpace 1 -routeClkNet –useCTSRouteGuide

-ocvAwareCluster is an important option if on chip variations are a concern. It will cluster registers to minimize the impact of on chip variation.

-usePowerInCts adds a constraint on the topology selection engine to minimize clock tree power. Turning this option on could impact the performance of the tree. This option will also enable the reporting of the clock tree power during clock tree synthesis (***ckSynthesis, clockDesign***) as well as clock reporting (***reportClockTree***). In addition the clock period needs to be specified in the clock tree definition file.

The remainder of the ***setCTSMode*** options are related to clock routing. The clocks will be routed by Nanoroute, and then if "***postOpt YES***" is used in the clock tree definition file, they will be optimized again for performance based on the final routes.

The user might want to consider issuing "***PadBufAfterGate YES***" in the clock tree definition file. This will favor insertion of padding buffers after a clock gating element thereby saving power when the clock gate is turned off. Note however that turning this option on sometimes causes clock gating violations.

The user should also be careful to avoid over constraining the skew requirements. An aggressive value could lead to the insertion of many buffers which would compromise power efficiency as well as the design sensitivity to on-chip variation.

Finally the choice of clock buffers and inverters is very important. The user should know whether to give priority to dynamic power savings or leakage savings. It is not always obvious what the best combination might be. Regardless of power some combinations might compromise the design performance targets as well as make signal integrity closure more difficult.

When three different VT threshold libraries are available there are a lot of potential options. Enabling high VT cells could lead to poor clock tree performance such as very long insertion delays, many buffers that could compromise dynamic power efficiency and design congestion. The leakage power from the clock tree however is likely to be good, even for a large number of buffers.

Enabling low VT cells will reduce the number of buffers and help design performance and congestion, but the leakage power could be significant.

It is also recommended to consider avoiding the mixing of very high drive strength buffers with very low drive strength. This could make signal integrity closure difficult. In any case the ***setCTSMode –preferredExtraSpace*** option should be used with a value of 1 or more.

7.2.6 Level Shifters Routing and Detail Routing

Before the signals can be routed, the level shifters power connections need to be completed. This step is only necessary for the VDDI pins of the LVLLH triple height level shifters. Sroute will ensure that straps are inserted every 50um to prevent IR-drop issues. Note that since we used ***setPlaceMode –maxShifterDepth 50*** during the flow the shifters on the left and right sides are placed close enough to the edge of the power domain to not require strapping. This was done so as to limit routing congestion. However on the bottom and top rows of the power domain, you will see the straps if any level shifters are used. Please check the **`connect_shifter_power.tcl`** script for more information.

Signals can then be routed using Nanoroute. Please refer to the **`init.tcl`** script for the full list of ***setNanoRouteMode*** options.

-setNanoRouteMode -routeWithTimingDriven true, ***-routeWithSiDriven true***, and ***-routeSiEffort high*** will prioritize timing sensitive routes while performing wire spreading to limit the amount of parasitic coupling between wires. These options will greatly help signal integrity closure. Wire spreading also helps achieve better yield.

setNanoRouteMode -quiet -drouteOptimizeUseMultiCutVia true, -drouteUseBiggerOverhangFirst true, and ***-drouteUseMultiCutViaEffort "high***" are essential for yield optimization. It will perform concurrent multicut via insertion. It is important to read section **4.2.3: generateLef command and multicut vias** about LEF optimization for multicut vias. With these optimizations the design can achieve close to 80% multicut via ratio, with all layers but via1 achieving between 90% and 100%. Because the number of via1 is so high in comparison to the other levels, special care must be taken to improve the via1 multicut definitions in the technology LEF.

7.2.7 VT-only optimization

Sometimes it might be beneficial to optimize timing by only using cell swapping between identical cells of different VT thresholds. The advantage is that since the physical footprints of the cells are identical there is no need for refine placement and eco routing.

It is only recommended to use this technique after routing optimization using ***optDesign –postRoute*** and when the design is very close to meeting timing (a few tens of picoseconds). The negative slack should be the result of the very last refine placement and eco routing calls of ***optDesign –postRoute***.

Using this technique in any other situation could greatly increase leakage power.

The alternative to using this technique when refine placement and eco routing are the cause of the negative slack is to increase ***setOptMode –setupTargetSlack***, however this could over optimize more paths than necessary if there is a large number of paths with around 0ns of setup slack.

7.2.8 Leakage Power Optimization

In Encounter 4.2 the timing optimization algorithms are free to use any cells available. Leakage doesn't have an associated cost function like timing or area. This concurrent leakage optimization feature will be available in an intermediate release of Encounter 5.2

In the meantime Leakage optimization is performed after post routing optimization. The design should be meeting timing before going into leakage optimization.

Internally Encounter will take a snapshot of all the cell types in the design and keep it in memory. Then it will swap all cells to their lowest leakage version (i.e. their high VT version). After that it will run timing analysis and gradually swap cells back to their original VT version on paths that are not meeting timing.

The command is ***optLeakagePower -postRoute -highEffort –checkTNS***

The ***–checkTNS*** option ensures that not only the worst negative slack (WNS) is not degraded but also the total negative slack (TNS) stays the same. Without this option the timing target is aligned to the WNS, so if the design is not meeting a lot of paths could be relaxed. Even when the WNS is close to 0ns Cadence recommends keeping the ***-checkTNS***. Without it some small degradation of the WNS could occur. Also if an early implementation iteration on the design is not meeting timing we still recommend keeping –***checkTNS*** active otherwise it will be difficult to accurately predict what the final leakage power number might be.

7.2.9 Hold Optimization

Hold optimization is performed using the optDesign command. It is important to run hold optimization after power leakage optimization. The reason for this is that many paths will be slowed down during leakage optimization when many cells are swapped to a lower VT version.

In order to limit the amount of leakage power drawn by hold fixing cells the user might want to consider disabling low VT and regular VT delay cells. In this example design MVH and MTH delay cells are disabled in the sdc constraint file. Maybe low VT and regular VT buffers could be disabled as well for this optimization only, with maybe the exception of the smallest MTH buffer in case it's needed to fix very small hold violations, and where a slower cell would impact setup.

To minimize hold violations on scan paths scan reordering can be run after clock tree synthesis using the ***setScanReorderMode –clkAware*** option.

Hold optimization is performed using the best case libraries and the RC best capacitance table. Please check the **`init.tcl`** script to see how this is achieved.

7.2.10 Signal Integrity Repair

Signal integrity (SI) repair is performed from within First Encounter using the Celtic noise analysis tool. Celtic is called directly by the ***optDesign –si*** command.

SI repair needs to be run on both the best case and the worst case corners. For this design it is recommended to run the worst case corner first.

In order to get an accurate noise analysis it is recommended to run power analysis for the corner being analyzed. The results can then be passed to the celtic engine using the ***setSIMode –irDropFiles {}*** option. The IR-drop results for all power and ground rails should be included.

Encounter will also automatically create a MSV file to pass on to Celtic. This file lists all cells and their associated nominal voltage. This is specific to a design with voltage islands and is done without user action. At the end of the implementation flow we will save these MSV files (one per corner) so they can be used in the signoff flow with Celtic NDC.

Note that the SI repair loop will fix noise glitches as well as any timing degradation due to crosstalk induced delays. The repair mechanism will attempt to recover the timing back to the worst negative slack value when noise delay is was not considered.

7.2.11 Decoupling Capacitance Insertion

There are several ways to insert decoupling capacitances in First Encounter. The most appropriate choice will depend on the design requirements for leakage power, as well as the placement congestion and iteration time.

For this flow the decoupling capacitance insertion is performed based on VoltageStorm generated eco files. In theory this means that we need to export the data required for VoltageStorm, run IR-drop in VoltageStorm, and then apply the eco files in First Encounter. This will provide the best accuracy and as a result the minimum amount of decoupling required to meet the design target, thus minimizing leakage from the decoupling capacitances. However for congested designs it might be difficult to add enough capacitances at this stage.

To avoid an interruption in the First Encounter scripts, we provide pre-generated Vstorm eco files. It is possible that doing so compromises the accuracy a little bit based on small placement and optimization variations from one run to another. See also sections **8.1.2: Decoupling Cap Insertion Techniques and Considerations**, and **8.1.5: VoltageStorm Dynamic (VSDG)** for further information.

If the design congestion doesn't allow the above methodology, then it's preferable to insert decoupling capacitances earlier in the flow, such as after placement. Another methodology could be to insert a matrix of decoupling caps spread evenly across the die as a base layer, thus reducing the amount of decoupling capacitances needed in the last stages.

Note that the Vstorm eco files provided have been modified as follows. The low and nominal VT decoupling capacitances have been commented out. Also the ***addDecapCell –powerDomain*** option was added with the power domain matching the rail for which the file was generated. Without this option there is a chance that decoupling capacitance does not get inserted in the intended power domain if the window under consideration straddles over two domains.

Finally the eco files need to be modified using the ***adjustDecapWindows.pl*** Perl script in order to avoid a problem in First Encounter when windows are created over region which do not belong to a power domain (IO pads for example). This problem was fixed, but could not be inserted in time for the First Encounter version targeted for this reference flow.

7.2.12 Final Data Creation

At the end of the implementation flow a number of files need to be created for use in the signoff flow. Once filler cells are inserted the design is complete. Note that we do not insert metal fills as this step is performed by the foundry. Any metal fill shapes present in the design would be discarded by the foundry and flagged as illegal during physical verification.

We generate the following files:

- logical verilog
- physical verilog for Conformal LP and LVS (though LVS is not part of this reference flow)
- full DEF file
- DEF file without routing information for use in Vstorm DG (info as to why in the signoff flow)
- Best RC and worst RC signoff Spef generated with the Fire and Ice QX extractor (from Encounter).
- MSV files for Celtic
- Timing window files for IR-drop analysis

8 SIGN-OFF FLOW

Prior to tape-out of a design, several key checks must be performed to insure the integrity and robustness of the physical implementation. The power grid should be checked for issues such as IR-drop effects, and power grid modifications and/or decoupling caps insertions implemented as necessary. Cross-coupling effects that can advance or retard signals, and introduce functional failures due to glitch noise must be identified and corrected. Logical equivalence checks should be run to insure that the functionality of the front-end netlist has been preserved throughout the physical implementation process. Finally, low-power designs must undergo additional functional and/or structural checks to insure correct level-shifting between different voltage domains, adequate isolation of switchable voltage islands, and proper power gating operation.

Some iteration between sign-off checking and physical implementation may be necessary until all effects have been corrected or mitigated and the design passes cleanly through all final sign-off checks. As with the implementation flow, it is assumed the user is familiar with standard sign-off checks and techniques, so this flow and user guide focuses primarily on those features unique to low power.

8.1 POWER INTEGRITY CHECKS

Voltage Storm static power-rail analysis identifies areas of excessive IR-drop on the power grid that can adversely affect circuit operation. Before moving on to dynamic power analysis, the power grid should be modified as necessary to eliminate any static IR-drop violations. Otherwise, excessive decoupling cap insertion may result, unnecessarily driving up power consumption on low-power designs.

After static power-rail analysis checks are free of IR-drop violations, VoltageStorm dynamic power analysis is then used to identify any additional transient IR-drop issues based on the actual switching activity of the design. This process creates ECO files that are fed back to the implementation flow to automatically insert the required decoupling caps needed to fix these violations. In addition, instance-based supply voltage data files are generated and fed forward to CeltIC NDC to allow the instance-specific effects of IR-drop to be accounted for during signal integrity analysis.

PowerMeter is run prior to power rail-analysis to generate the instance-based power consumption data required for static analysis, and the dynamic PWL average tap current data needed for dynamic analysis. Voltage source locations needed for both static and dynamic analysis are generated by First Encounter during implementation.

8.1.1 Setting Voltages and IR-drop Limits

Voltage settings and IR-drop limits must be applied consistently throughout both the power integrity and signal integrity flows. Careful selection of these settings assures adequate power supply for proper circuit operation, while allowing the designer to take maximum advantage of design techniques to reduce power. Excessive conservatism will be counterproductive to achieving maximum power reduction.

Based on representative production designs, the sign-off voltages chosen for the reference flow are based on an expected maximum off-chip variation of 2.5% per rail (5% rail-to-rail). For static power-rail analysis, the core IR-drop limit is held to 2.5% per rail (5% rail-to-rail). The core IR-drop limits for dynamic analysis are increased to 5% per rail (10% rail-to-rail).

Examine the appropriate scripts to see how these settings translate into parameters for the analyze commands in the VoltageStorm runs. Since VSS is held to 0V, VDD voltages are set to reflect the full rail-to-rail variation. IR-drop limits are set individually for VSS and VDD, so these values reflect per-rail variation.

See section **8.1.5: VoltageStorm Dynamic (VSDG)**, for a specific example of the dynamic voltage and IR-drop limits.

Note that the voltage and IR-drop limit settings used for the reference flow design are suggested as representative guidelines only. These must be verified and/or modified based on knowledge of voltage source tolerances and other considerations unique to each design.

8.1.2 Decoupling Cap Insertion Techniques and Considerations

As mentioned in section **7.2.11: Decoupling Capacitance Insertion**, various techniques may be employed for decoupling cap insertion based on the requirements and constraints of a particular design. For low-power designs, the goal is, of course, to insert the minimum amount of decoupling caps to meet design requirements. While in theory all decoupling caps may be inserted after placement, routing, and optimization are complete, design congestion and other factors may make this difficult or impossible.

Adding some decoupling caps early in the implementation may mitigate these issues. Often a checkerboard pattern of decoupling caps is added prior to placement. Such strategies must be employed judiciously, since they can be counterproductive to achieving the lowest possible power consumption by adding decoupling caps where they are not actually needed. Placing decoupling caps close to other cells that are likely to need them, such as clock buffers, is another possible strategy to consider. Custom scripts may be written to identify such cells and place decoupling caps next to them. In the case of clock buffers, it may be advantageous to create a special clock buffer cell that includes both the buffer itself and a decoupling cap, and use this specialized cell for clock tree design.

On the Osprey design, an early run of Voltage Storm dynamic analysis was used to generate an initial ECO file to insert decoupling caps into the design. The power integrity checks in this reference flow section are designed to demonstrate how Voltage Storm identifies the need for a small amount of additional decoupling cap insertion in some areas. This allows the user to see examples of actual decoupling cap ECO files. As mentioned, this can be an iterative process, and in a production environment, these additional decaps would need to be inserted into the design and another pass of sign-off checks would need to be run to verify the results.

8.1.3 PowerMeter

PowerMeter is used to generate both the instance-based power consumption data and average tap current data needed during static and dynamic power-rail analyses, respectively.

```
>% cd WORK/POWER_INTEGRITY
>% source ../../SCRIPTS/SETUP_POWER_INTEGRITY.csh
>% source ../../SCRIPTS/pm_bc.csh
>% source ../../SCRIPTS/pm_wc.csh
```

The BC and WC PowerMeter runs can be executed in parallel. The subsequent VoltageStorm runs require data from PowerMeter, so these PowerMeter runs must complete successfully before proceeding.

BC output data is located at **`../../DATA/POWER_INTEGRITY/POWERMETER/bc`**.
WC output data is located at **`../../DATA/POWER_INTEGRITY/POWERMETER/wc`**.

`powermeter.log`	log file
`powermeter.pwr`	instance-based power consumption (used in static power-rail analysis)
`dynamic_VDD_hi_power.ptiavg` **`dynamic_VDD_lo_power.ptiavg`** **`dynamic_VSS.ptiavg`**	average tap current data (used in dynamic power-rail analysis)

The section **13.1: PowerMeter Warnings**, for a list of expected warning messages.

8.1.4 VoltageStorm Static (VSPE)

VoltageStorm static power-rail analysis may be performed only after PowerMeter runs have been successfully completed.

```
>% cd WORK/POWER_INTEGRITY
>% source ../../SCRIPTS/SETUP_POWER_INTEGRITY.csh
```

Analysis is done on a per-rail basis, so there are six (6) dynamic runs to execute, which can be run in parallel:

```
>% source ../../SCRIPTS/vstorm2_static_hi_bc.csh
>% source ../../SCRIPTS/vstorm2_static_hi_wc.csh
>% source ../../SCRIPTS/vstorm2_static_lo_bc.csh
>% source ../../SCRIPTS/vstorm2_static_lo_wc.csh
>% source ../../SCRIPTS/vstorm2_static_vss_bc.csh
>% source ../../SCRIPTS/vstorm2_static_vss_wc.csh
```

Results are located at: **`../../DATA/POWER_INTEGRITY/VSPE`**

Six subdirectories hold the output for each of the six rail/corner cases:

`VDD_hi_bc/`	**`VDD_hi_wc/`**
`VDD_lo_bc/`	**`VDD_lo_wc/`**
`VSS_bc/`	**`VSS_wc/`**

The following table lists the names and descriptions of key files for the **VDD_hi_bc** case. Similar names/descriptions apply to the other five cases.

VDD_hi_power_-40C_avg_1	Subdirectory that, among other things, contains GIF files for all the layers. These can be opened to graphically view the power grid layer-by-layer.
static_iv_vdd_hi_bc.txt	Instance-based voltage supply file. Since the reference flow takes advantage of the more accurate dynamic analysis as well, these files are not used for CeltIC; instead their dynamic equivalents from VoltageStorm dynamic are used to pass along IR-drop information to the signal integrity flow.
vstorm_static_vdd_hi_bc.log	Log file.
zx.log	Log file generated when zx called from within VoltageStorm to perform extraction.

Examine each of the vstorm log files in turn to confirm that none of the static IR-drop limits were violated in these runs. (Had there been any violations, they would need to be addressed before moving on to dynamic power-rail analysis.

See section **13.2: VoltageStorm Static Warnings**, for a list of expected warning messages.

8.1.5 VoltageStorm Dynamic (VSDG)

VoltageStorm dynamic power-rail analysis may be performed only after PowerMeter runs have been successfully completed, and all static IR-drop violations reported during static power-rail analysis have been eliminated.

>% cd WORK/POWER_INTEGRITY
>% source ../../SCRIPTS/SETUP_POWER_INTEGRITY.csh

Analysis is done on a per-rail basis, so there are six (6) dynamic runs to execute, which can be run in parallel:

>% source ../../SCRIPTS/vstorm2_dynamic_hi_bc.csh
>% source ../../SCRIPTS/vstorm2_dynamic_hi_wc.csh
>% source ../../SCRIPTS/vstorm2_dynamic_lo_bc.csh
>% source ../../SCRIPTS/vstorm2_dynamic_lo_wc.csh
>% source ../../SCRIPTS/vstorm2_dynamic_vss_bc.csh
>% source ../../SCRIPTS/vstorm2_dynamic_vss_wc.csh

Results are located at: **../../DATA/POWER_INTEGRITY/VSDG**

Six subdirectories hold the output for each of the six rail/corner cases:

VDD_hi_bc/	**VDD_hi_wc/**
VDD_lo_bc/	**VDD_lo_wc/**
VSS_bc/	**VSS_wc/**

The following table lists the names and descriptions of key files for the **VDD_hi_bc** case. Similar names/descriptions apply to the other five cases.

VDD_hi_power_-40C_dynamic_1/	Subdirectory that, among other things, contains GIF files for all the layers. These can be opened to graphically view the power grid layer-by-layer.
dynamic_iv_vdd_hi_bc.txt	Instance-based voltage supply file. These files are passed forward to CeltIC NDC so that instance-specific IR-drop effects can be taken into account during the signal integrity flow.
vstorm_dynamic_vdd_hi_bc.log	Log file.
zx.log	Log file generated when zx called from within VoltageStorm to perform extraction.

Examining each of the six VoltageStorm log files in turn will show that minor violations of the dynamic IR-drop limits show up in the bc analysis on both power rails, while there are no violations in the wc for either power rail.

For example, the bc analysis for the high voltage rail, VDD_hi, is run at a voltage of 1.260V (5% rail-to-rail increase above the nominal 1.2V for this rail). The allowed IR-drop is 2.5%, per rail, of the nominal voltage, or 60mV. Therefore, the IR-drop voltage limit is 1.200V (1.260V – 0.060V). From the log file (**VDD_hi_bc/vstorm_dynamic_vdd_hi_bc.log**) the actual maximum IR-drop is found to be 1.170V; and the total number of violations is 64**:**

```
Data filtering results for IR-drop:
        Overall data range:   1.170V - 1.260V
        Overall data average: 1.221V

Filter 1:  8188699  of  8188699  data values fell into this filter.
        filtered data range:   1.170V - 1.260V
        filtered data average: 1.221V
              64 values were in range 1: 1.170V - 1.200V
           11312 values were in range 2: 1.200V - 1.205V
         1909110 values were in range 3: 1.205V - 1.210V
         1594055 values were in range 4: 1.210V - 1.215V
         1472868 values were in range 5: 1.215V - 1.220V
          863613 values were in range 6: 1.220V - 1.225V
          573560 values were in range 7: 1.225V - 1.230V
         1764117 values were in range 8: 1.230V - 1.260V
```

In addition to flagging these violations, VoltageStorm creates an ECO file indicating the amount of decoupling cap required in one or more windows to smooth out the IR-drop, along with a listing of potential decap cell types and their corresponding decap values. This can be read into First Encounter, which will attempt to automatically place these decoupling caps (subject to constraints of local congestion).

For this bc VDD_hi case, the ECO file (**VDD_hi_bc/vdd_hi_bc.eco**) indicates the need for about 28pF of *additional* decoupling caps sprinkled over 8 windows (regions) to bring the dynamic IR-drop for this rail (under bc conditions) back within the IR-drop limit. Note that about 6pF of decoupling caps had already been inserted (as discussed in section **8.1.2: Decoupling Cap Insertion Techniques and Considerations**):

```
# define decap cell candidates
addDeCapCellCandidates FILLCAP16MHT 169.244
addDeCapCellCandidates FILLCAP32MHT 354
…
addDeCapCellCandidates FILLCAP32MVH 178.174

# addDeCap -totCap value_in_fF -area x1_in_um y1_in_um x2_in_um y2_in_um
addDeCap -totCap 83.0177 -area 600 800 800 1000
addDeCap -totCap 665.456 -area 2000 1600 2200 1800
addDeCap -totCap 7601.44 -area 1800 1800 2000 2000
addDeCap -totCap 4379.3 -area 2000 1800 2200 2000
addDeCap -totCap 6571.77 -area 1600 2000 1800 2200
addDeCap -totCap 910.04 -area 1000 2200 1200 2400
addDeCap -totCap 49.3195 -area 1000 2400 1200 2600
addDeCap -totCap 20249.6 -area 2000 2400 2200 2600
# Total decoupling capacitances added with addDecap: 34153.5 (fF)
# Total decoupling capacitances already available:   6356.49 (fF)
```

As mentioned in section **8.1.2: Decoupling Cap Insertion Techniques and Considerations**, these small violations were not fixed to help the user become familiar with how Voltage Storm dynamic analysis identifies and reports IR-drop violations, and how it provides an automatic fixing mechanism to feed back to the implementation flow. In a production environment, these few remaining violations would need to be addressed prior to taping out the design.

Notice that there are also very small (approximately 2.5mV) violations of the dynamic IR-drop limits for the VSS rail (both conditions). However, no decoupling caps are called for in those ECO files. In this case, the violations are small enough to not warrant adding decoupling caps. Examining the bc VSS case, for example, (**VSS_wc/vstorm_dynamic_vss_wc.log**) shows that the small violations do not require decoupling caps:

```
Data filtering results for decoupling cap requested for IR fix:
        Overall data range:   0.000F - 0.000F
        Overall data average: 0.000F
```

SignalStorm dynamic analysis also generates instance-based voltage supply files for each rail/condition. This allows CeltIC NDC to take into account instance-specific IR-drop effects during signal integrity analysis.

See section **13.3: VoltageStorm Dynamic Warnings,** for a list of expected warning messages.

8.2 SIGNAL INTEGRITY CHECKS

Prior to release for tape-out, nanometer-class designs must be checked for functional failures due to crosstalk glitch noise, and for crosstalk-induced delay effects. CeltIC™ NDC is run on the reference flow design to detect a handful of remaining glitches, and to generate incremental SDF representing crosstalk-induced delay, both in an IR-drop aware context, using effective current source models (ECSM) and transistor modeling for fast, accurate analysis. In a production environment, these glitches would have to be fixed, and the incremental SDF would also be fed back to the First Encounter implementation flow so that critical crosstalk-induced delays could be rectified.

To accomplish this, CeltIC NDC requires parasitic information (spef) generated during physical implementation. And, since this design has multiple power domains, instance-based voltage domain assignments (msmv file) are also required (also generated during physical implementation). To account for IR-drop, CeltIC NDC also needs an instance-based voltage supply (a.k.a. “IR-drop”) file for each rail, generated during dynamic power-rail analysis.

Care must be taken to insure that the same voltage levels used during power integrity analysis are also used during signal integrity analysis. Using carefully selected expected voltage levels (see section **8.1.1: Setting Voltages and IR-drop Limits**), combined with accurate ECSM cell models and the inclusion of dynamic IR-drop effects, allows the designer to take maximum advantage of power-reduction techniques without fear of compromising the design's signal integrity.

Important Note (please refer to section **1.3: Tools and Tool Versions**): A different version of Celtic than the one packaged with SOC42USR5 is required for this flow to work properly. To insure that the correct version of Celtic is called during signal integrity analysis, the user needs to update the CELTIC_PATH environment variable in the **`SETUP_SIGNOFF_SI.csh`** script to reflect the location of this binary in the local installation before starting the flow.

>% cd WORK/SIGNOFF_SI
>% source ../../SCRIPTS/SETUP_SIGNOFF_SI.csh
>% source ../../SCRIPTS/signoff_si_checks_prep.csh

The wc and bc signal integrity checks can be run in parallel:

>% source ../../SCRIPTS/signoff_si_checks_bc.csh
>% source ../../SCRIPTS/signoff_si_checks_wc.csh

Results of the celtic_ndc runs can be found in the directory **`../../DATA/SIGNOFF_SI.`**

bc.log wc.log	CeltIC log files
bc_incremental_signoff.sdf wc_incremental_signoff.sdf	Crosstalk-induced delay effects; can be fed back to the implementation flow for any required fixing.
bc_rcvr_peak_slack_failure.text wc_rcvr_peak_slack_failure.text	Crosstalk-induced glitches.

Examination of the incremental SDF files shows some crosstalk-induced delay remains for both the bc and wc analyses. While there are no glitch violations for wc, there are 3 remaining glitches uncovered in the bc analysis: One glitch involving a victim net driving a sequential element, and two involving the same pair of victim nets driving a single common combinatorial gate. Note that the glitch threshold is 25% of the rail voltage for sequential elements, and 80% of the rail voltage for combinatorial gates. So in this case, all 3 glitches involve the default low power (VDD_lo rail) domain.

See section **13.4: CeltIC NDC Warnings** , for a list of expected warning messages.

8.3 LOGICAL EQUIVALENCY CHECKS

Encounter™ Conformal ASIC EC is used to perform standard logical equivalency checks (LEC) between the "golden" front-end gate-level verilog netlist and the final "revised" backend verilog netlist to insure that no logic changes have been inadvertently introduced during implementation. In addition, further structural and functional checks may need to be performed on low-power designs (see section **8.4, Low Power Checks**).

To run the logical equivalency sign-off checks, do the following:

>% cd WORK/LOGIC_EQV_CHECK
>% source ../../SCRIPTS/SETUP_LOGIC_EQV_CHECK.csh
>% source ../../SCRIPTS/logic_eqv_check.csh

The resultant logfile is **`../../DATA/LOGIC_EQV_CHECK/lec.log`**.

The following classes of library related infos/warnings do not affect the results and can be ignored:

// Warning: (RTL7.5) Input signal assigned by logic
// Warning: (RTL14) Signal with fanin drive and no fanout load is removed
// Warning: (DIR6.1) Compiler directive ignored
// Warning: (DIR6.2) Compiler directive supported
// Warning: (IGN3.2) Module/entity duplicated and later ones ignored
// Note: (HRC1.2) Module/entity not translated (black-boxed)
// Warning: (HRC3.8) Port positional association exist in instantiation

The following classes of design-related infos/warning do not affect the results and can be ignored:

// Warning: (RTL19.1) Identifier is a reserved keyword and may conflict in mixed languages design
// Warning: (HRC3.5a) Open input/inout port connection detected
// Note: (HRC3.5b) Open output port connection detected

8.4 LOW POWER CHECKS

For designs that incorporate multiple core voltages and/or power shut-off techniques to reduce power consumption, standard logical equivalency checking is not sufficient to fully insure design integrity and functionality. Instead, these designs require additional structural and/or functional checks to insure correct level-shifting between different voltage domains, adequate isolation of switchable voltage islands, and proper power gating operation.

Encounter Conformal Low Power verifies correct implementation of these low power design techniques and allows validation of silicon using formal techniques (versus simulation) early in the design process. It also decreases the risk of bugs which are often missed, before a product goes out the door.

8.4.1 Sign-Off Low Power Checks on Physical Verilog Netlist

Whereas LEC is run to compare two versions of a netlist against each other, Conformal Low Power checks run on a single netlist to determine whether the implementation of low power mechanisms employed on the design are structurally and functionally sound. Potential problems involve level shifters placed in an incorrect power domain, isolation cells missing between switchable power islands, and improper power gating.

To perform low power checks, the user must specify the top-level power/ground pins, define domains for the various voltage islands, and specify special cells such as level-shifters, isolation cells, and power gating/shut-off cells.

The Osprey design has two “always on” (non-switchable) voltage domains, operating at 1.0V and 1.2V nominal, respectively. Conformal Low Power structural checks verify that the necessary level-shifter cells are inserted on signals that communicate across different voltage boundaries, and that they are placed in the proper power domains. Since both voltage domains are “always on”, isolation and power gating checks are not required for the Osprey design.

For sign-off, Conformal Low Power structural and functional checks should be performed on the final "physical" gate-level netlist output by SoC Encounter at the end of the implementation flow. This physical verilog should correspond to the final revised backend "logical" verilog netlist used for LEC. The physical verilog netlist contains the same information as its logical counterpart, with the addition of power and ground connectivity information derived from the actual physical implementation database.

To run the Conformal Low Power sign-off checks, do the following:

```
>% cd WORK/LOW_POWER_CHECKS
>% source ../../SCRIPTS/SETUP_LOW_POWER_CHECKS.csh
>% source ../../SCRIPTS/low_power_checks.csh
```

The resultant log file is **`../../DATA/LOW_POWER_CHECKS/clp.physical.log`**.

Note that there are no warnings/errors after the ***analyze power domain*** step. The script can be run with Conformal in the "gui" mode if the user wishes to graphically view the low-power rule check results and/or explore the netlist schematic to see how Conformal assigned each net and component to given voltage and power domains.

8.4.2 Early Low Power Checks Using Logical Verilog Netlist (Optional)

Conformal Low Power has the ability to run structural checks on a *logical* netlist as well. This capability is useful, for instance, in double-checking level-shifting and/or isolation in the front-end netlist prior to beginning back-end implementation. The flow is very similar to low power checks with a physical netlist. However, since a logical netlist does not contain power connectivity information, this must be explicitly specified by the user in the Conformal Low Power dofile. (When using a physical netlist, the true power connectivity is extracted from the actual physical implementation database and added to the verilog netlist.)

To demonstrate this capability, and how it differs from using the physical netlist, a dofile is provided to run on the released front-end logical netlist. In practice, this should be run prior to beginning the backend implementation phase. To run this optional check, do the following:

```
>% cd WORK/LOW_POWER_CHECKS
>% source ../../SCRIPTS/SETUP_LOW_POWER_CHECKS.csh
>% source ../../SCRIPTS/low_power_logical_checks.csh
```

The resultant log file is **`../../DATA/LOW_POWER_CHECKS/clp.logical.log`**.

Note that there are no warnings/errors after the ***check lowpower structure*** step. The script can be run with Conformal in the "gui" mode if the user wishes to graphically view the low-power rule check results and/or explore the netlist schematic to see how Conformal assigned each net and component to given voltage and power domains.

Important Note: For sign-off, Conformal Low Power checks should only be performed on a physical verilog netlist, which contains the actual power/ground connectivity information derived from the physical implementation database. This allows Conformal to detect level-shifters and/or isolation cells that might be placed in incorrect power domains, by virtue of the resulting incorrect power connections. When using a logical netlist for low power checking, Conformal must rely upon the power connectivity information provided by the user, as opposed the actual power connectivity. If the implementation does not match the assumed power connectivity, structural problems may be missed.

9 BRIDGING FAULTS ANALYSIS

Movement to deep submicron (nanometer) processes introduces the possibility of many new process-related failure mechanisms. Chief among these are defects between metal lines arising during manufacturing.

A detailed application note on Cadence Design System's "SourceLink" provides background on this subject, and explains the unique capabilities in the Cadence Encounter Test and Diagnostics suite of tools to address many of these issues. The application note can be found at:

http://sourcelink.cadence.com/docs/files/Application_Notes/2005/dftBridgingFaultsAN.pdf

For the Osprey design, a file containing bridging fault candidate net pairs were extracted from the SoC Encounter database. These candidate net pairs were then read into Encounter Test, where the **create_shorted_net_faults** command was used to produce a file of bridging fault specifications covering a variety of conditions for each pair. These are then represented in the set of faults to be targeted for test, and defect bridging tests were created in the same way as other static and dynamic tests. (See application note for more details.)

The bridging faults candidates file, run scripts, the resultant vectors, and a copy of the SourceLink application note are provided as part of this reference flow tarkit:

>% cd SOURCE/BRIDGING_FAULTS

README	
bridging_faults.meth	contains all the ET commands required to build the model and generate WGL vectors
dftBridgingFaultsAN.pdf	Bridging Faults application note (copy of PDF from Cadence SourceLink)
et_inputs/	net pairs and other required input files
library/	library data required for the flow
netlists/	input netlist
reports/	report files
wgl/	resultant vectors

10 APPENDIX A: LIBRARY CONTENTS

Timing Libraries (BC)	Cells	Vth
scmetro_cms9flp_hvt_ff_1p1v_m40c.lib	STD CELL	HVT
scmetro_cms9flp_hvt_ff_1p32v_m40c.lib	STD CELL	HVT
scmetromvk_cms9flp_hvt_ff_1p1v_1p32v_m40c.lib	MVK	HVT
scmetro_cms9flp_rvt_ff_1p1v_m40c.lib	STD CELL	RVT
scmetro_cms9flp_rvt_ff_1p32v_m40c.lib	STD CELL	RVT
scmetromvk_cms9flp_rvt_ff_1p1v_1p32v_m40c.lib	MVK	RVT
scmetro_cms9flp_lvt_ff_1p1v_m40c.lib	STD CELL	LVT
scmetro_cms9flp_lvt_ff_1p32v_m40c.lib	STD CELL	LVT
scmetromvk_cms9flp_lvt_ff_1p1v_1p32v_m40c.lib	MVK	LVT
iometrost_cms9flp_hvt_ff_1p1v_2p7v_m40c.lib	IO	HVT
iometrost_cms9flp_hvt_ff_1p32v_2p7v_m40c.lib	IO	HVT
RAM2P_1024x32_ff_1v10_m40c_syn.lib	RAM	HVT
RAM2P_1024x32_ff_1v32_m40c_syn.lib	RAM	HVT
RAM2P_128x16_ff_1v10_m40c_syn.lib	RAM	HVT
RAM2P_128x16_ff_1v32_m40c_syn.lib	RAM	HVT
RAM_1024x32_ff_1v10_m40c_syn.lib	RAM	HVT
RAM_1024x32_ff_1v32_m40c_syn.lib	RAM	HVT
RAM_4096x32_ff_1v10_m40c_syn.lib	RAM	HVT
RAM_4096x32_ff_1v32_m40c_syn.lib	RAM	HVT
Timing Libraries (WC)	**Cells**	**Vth**
scmetro_cms9flp_hvt_ss_0p9v_125c.lib	STD CELL	HVT
scmetro_cms9flp_hvt_ss_1p08v_125c.lib	STD CELL	HVT
scmetromvk_cms9flp_hvt_ss_0p9v_1p08v_125c.lib	MVK	HVT
scmetro_cms9flp_rvt_ss_0p9v_125c.lib	STD CELL	RVT
scmetro_cms9flp_rvt_ss_1p08v_125c.lib	STD CELL	RVT
scmetromvk_cms9flp_rvt_ss_0p9v_1p08v_125c.lib	MVK	RVT
scmetro_cms9flp_lvt_ss_0p9v_125c.lib	STD CELL	LVT
scmetro_cms9flp_lvt_ss_1p08v_125c.lib	STD CELL	LVT
scmetromvk_cms9flp_lvt_ss_0p9v_1p08v_125c.lib	MVK	LVT
iometrost_cms9flp_hvt_ss_0p9v_2p3v_125c.lib	IO	HVT
iometrost_cms9flp_hvt_ss_1p08v_2p3v_125c.lib	IO	HVT
RAM2P_1024x32_ss_0v90_125c_syn.lib	RAM	HVT
RAM2P_1024x32_ss_1v08_125c_syn.lib	RAM	HVT
RAM2P_128x16_ss_0v90_125c_syn.lib	RAM	HVT
RAM2P_128x16_ss_1v08_125c_syn.lib	RAM	HVT
RAM_1024x32_ss_0v90_125c_syn.lib	RAM	HVT
RAM_1024x32_ss_1v08_125c_syn.lib	RAM	HVT
RAM_4096x32_ss_0v90_125c_syn.lib	RAM	HVT
RAM_4096x32_ss_1v08_125c_syn.lib	RAM	HVT

Table 2: Timing (.lib) Views

CeltIC Signal Integrity (BC)	Cells	Vth
cms9flphvt_m.lpeSpc_ff_1.32V_1.1V_-40C.CDB	STD CELL	HVT
cms9flphvt_m.lpeSpc_lvl_1.32VDDV_1.1VDDI_-40C_ff.CDB	LH MVK	HVT
cms9flphvt_m.lpeSpc_lvl_LVLHL_1.1V_-40C_ff.CDB	HL MVK	HVT
cms9flprvt_m.lpeSpc_ff_1.32V_1.1V_-40C.CDB	STD CELL	RVT
cms9flprvt_m.lpeSpc_lvl_1.32VDDV_1.1VDDI_-40C_ff.CDB	LH MVK	RVT
cms9flprvt_m.lpeSpc_lvl_LVLHL_1.1V_-40C_ff.CDB	HL MVK	RVT
cms9flplvt_m.lpeSpc_ff_1.32V_1.1V_-40C.CDB	STD CELL	LVT
cms9flplvt_m.lpeSpc_lvl_1.32VDDV_1.1VDDI_-40C_ff.CDB	LH MVK	LVT
cms9flplvt_m.lpeSpc_lvl_LVLHL_1.1V_-40C_ff.CDB	HL MVK	LVT
cms9flphvt_m.lpeSpc_io_2.7DVDD_1.32_and_1.1_VDD_-40C_ff.CDB	IO	HVT
RAM2P_1024x32_ff_1.32V_1.1V_-40C.CDB	RAM	HVT
RAM2P_256x16_ff_1.32V_1.1V_-40C.CDB	RAM	HVT
RAM_1024x32_ff_1.32V_1.1V_-40C.CDB	RAM	HVT
RAM_4096x32_ff_1.32V_1.1V_-40C.CDB	RAM	HVT
CeltIC Signal Integrity (WC)	**Cells**	**Vth**
cms9flphvt_m.lpeSpc_ss_1.08V_0.9V_125C.CDB	STD CELL	HVT
cms9flphvt_m.lpeSpc_lvl_1.08VDDV_0.9VDDI_125C_ss.CDB	LH MVK	HVT
cms9flphvt_m.lpeSpc_lvl_LVLHL_0.9V_125C_ss.CDB	HL MVK	HVT
cms9flprvt_m.lpeSpc_ss_1.08V_0.9V_125C.CDB	STD CELL	RVT
cms9flprvt_m.lpeSpc_lvl_1.08VDDV_0.9VDDI_125C_ss.CDB	LH MVK	RVT
cms9flprvt_m.lpeSpc_lvl_LVLHL_0.9V_125C_ss.CDB	HL MVK	RVT
cms9flplvt_m.lpeSpc_ss_1.08V_0.9V_125C.CDB	STD CELL	LVT
cms9flplvt_m.lpeSpc_lvl_1.08VDDV_0.9VDDI_125C_ss.CDB	LH MVK	LVT
cms9flplvt_m.lpeSpc_lvl_LVLHL_0.9V_125C_ss.CDB	HL MVK	LVT
cms9flphvt_m.lpeSpc_io_2.3DVDD_1.08_and_0.9_VDD_125C_ss.CDB	IO	HVT
RAM2P_1024x32_ss_1.08V_0.9V_125C.CDB	RAM	HVT
RAM2P_256x16_ss_1.08V_0.9V_125C.CDB	RAM	HVT
RAM_1024x32_ss_1.08V_0.9V_125C.CDB	RAM	HVT
RAM_4096x32_ss_1.08V_0.9V_125C.CDB	RAM	HVT

Table 3: Signal Integrity/Noise (Celtic .cdb) Views

VoltageStorm Power Analysis Libraries	Cells	Vth
IBM_HVT_IO.cl	IO	HVT
IBM_HVT_RAM2P_1024_32.cl	RAM	HVT
IBM_HVT_RAM2P_128_16.cl	RAM	HVT
IBM_HVT_RAM_1024_32.cl	RAM	HVT
IBM_HVT_RAM_4096_32.cl	RAM	HVT
IBM_HVT_SC_DV.cl	STD CELL	HVT
IBM_RVT_DV.cl	STD CELL	RVT
IBM_LVT_DV.cl	STD CELL	LVT

Table 4: Power Analysis (Vstorm .cl) Views

SignalStorm Libraries	Cells	Vth
hvt_ff_1.1.lib	ALL	HVT
hvt_ff_1.32.lib	ALL	HVT
rvt_ff_1.1.lib	ALL	RVT
rvt_ff_1.32.lib	ALL	RVT
lvt_ff_1.1.lib	ALL	LVT
lvt_ff_1.32.lib	ALL	LVT
hvt_ss_0.9.lib	ALL	HVT
hvt_ss_1.08.lib	ALL	HVT
rvt_ss_0.9.lib	ALL	RVT
rvt_ss_1.08.lib	ALL	RVT
lvt_ss_0.9.lib	ALL	LVT
lvt_ss_1.08.lib	ALL	LVT

Table 5: Signal Storm (.lib with ECSM) Views

LEF P&R Abstract Views	Cells	Vth
cms9flp_8lm_2thick_tech.optimized.lef	TECH	
cms9flphvt_lvl_macros.lef	MVK	HVT
cms9flphvt_m_macros.lef	STD CELL	HVT
cms9flprvt_lvl_macros.lef	MVK	RVT
cms9flprvt_m_macros.lef	STD CELL	RVT
cms9flplvt_lvl_macros.lef	MVK	LVT
cms9flplvt_m_macros.lef	STD CELL	LVT
iometrost_cms9flp_hvt_M6_02_00.lef	IO	HVT
RAM2P_1024x32.vclef	RAM	HVT
RAM2P_128x16.vclef	RAM	HVT
RAM_1024x32.vclef	RAM	HVT
RAM_4096x32.vclef	RAM	HVT

Table 6: P&R Abstract (LEF) Views

11 APPENDIX B: REFERENCES

- What's New in CeltIC™NDC, Product Version 5.2.1
- CeltIC™ NDC User Guide, Product Version 5.2.1
- CeltIC NDC™ Command Reference, Product Version 5.2.1

- Analysis Product Notes: What's New in ANLS6.1, Product Version ANLS6.1
- VoltageStorm PowerMeter Manual, March 2006
- VoltageStorm Transistor-Level Rail Analysis User Guide, March 2006
- VoltageStorm Cell-Level Rail Analysis Reference Manual, March 2006

- Fire & Ice QXC User Guide, Product Version 3.4
- Fire & Ice QXC Command Reference Manual

- What's New in Encounter™ Conformal, Product Version 5.2
- Encounter™ Conformal ASIC Extended Checks Reference Manual, Product Version 5.2
- Encounter™ Conformal Equivalence Checking Reference Manual, Product Version 5.2
- Encounter™ Conformal Equivalence Checking User Guide, Product Version 5.2
- Encounter™ Conformal Equivalence Checking Quick Reference, Product Version 5.2

- What's New in Encounter ™ 4.2.4, Product Version 4.2.4
- Encounter™ User Guide, Product Version 4.2.4
- Encounter™ Text Command Reference, Product Version 4.2.4
- Encounter™ Menu Reference, Product Version 4.2.4

- Incisive Comprehensive Coverage Quick Reference Guide, Product Version 5.5
- Product Documentation for IUS 5.5
- Command Line Guide, Product Version 3.0

- Encounter™ Test Design Concepts and Applications, Product Version 3.0
- Encounter™ Test Graphical User Interface Reference, Product Version 3.0
- Automating DFT Integration in the Top Shell Application Note, Product Version 2.2 December 2005
- Memory Built-In Self Test Reference, Product Version 3.0
- Encounter™ Test Library Data Reference, Product Version 3.0
- Encounter™ Diagnostics Tutorial, Product Version 3.0

- What's New in Encounter™ RTL Compiler, Product Version 5.2
- Quick Reference for Encounter™ RTL Compiler , Product Version 5.2
- Command Reference for Encounter™ RTL Compiler, Product Version 5.2.2
- Design For Test in Encounter ™ RTL Compiler, Product Version 5.2.2
- Low Power in Encounter™ RTL Compiler, Product Version 5.2.2
- Setting Constraints and Performing Timing Analysis Using Encounter™ RTL Compiler , Product Version 5.2.2
- Using Encounter™ RTL Compiler, Product Version 5.2.2
- Interfacing Between RTL Compiler and Conformal LEC, Product Version 5.2.2
- GUI Guide for Encounter™ RTL Compiler , Product Version 5.2.2

12 APPENDIX C: GLOSSARY

AHB: Advanced High-performance Bus. Part of the AMBA specification. Osprey design uses AMBA/AHB bus.

AMBA: Advance Microcontroller Bus Architecture (developed by ARM). Osprey design uses AMBA/AHB bus.

Isolation Cell: Implementation of switchable voltage islands require isolated from other domains by "isolation cells" — latches that engage to prevent transients or indeterminate states from a powered-down cell from reaching the inputs of an active cell. Prevents leakage caused by unknown state outputs from the shutdown area. See also Level Shifter.

LEC: Logical Equivalency Checking.

Level Shifter: Mixing of voltages on chip requires the use of level-shifter cells that translate between signal levels. Cell used for connecting signals between different power domains. See also Isolation Cell.

MSV: (Multi Supply Voltage).

MSSV: (Multi Supply Single Voltage): Core runs at single voltage, but some portions of the logic are isolated on their own power supply.

MSMV: (Multi Supply Multi Voltage): Supplies of different voltages are used for the core logic.

MTCMOS: Multiple Threshold CMOS. Standard cell libraries that combine these high and low threshold devices using power switches are called multiple threshold CMOS (MTCMOS) libraries.

Power Gating: A power gated design uses power switches made from high voltage threshold (Vt) transistors to effectively "turn off" the connection to power or ground for low Vt standard cells, and thus "turn off" leakage power when the design is in "stand-by" mode. Power gating achieves faster design speeds associated with high performance standard cell libraries while maintaining the longer battery life of low power libraries. Standard cell libraries that combine these high and low threshold devices using power switches are called multiple threshold CMOS (MTCMOS) libraries. See also Power Switching.

Power Domain (a.k.a. Voltage Island): The concept of a power domain is central to MSV design with the SoC Encounter™ platform. A power domain (also known as a voltage island) is a floorplan object in the Encounter software. A power domain has a fence constraint, and a specific library (.lib, .lef) bound to it. Cells that belong to a power domain can be placed only within that power domain. The exception to this rule is level shifter cells, which are used to communicate between power domains of different voltages. By constraining the design this way, a complete place and route flow can be used on an MSV design. You can automatically place level shifters (even if they are not in your netlist), perform timing optimization, run clock tree synthesis (CTS) across domain boundaries, and obtain DRC clean power routing.

Power Switching: See Power Gating.

13 APPENDIX D: SIGN-OFF FLOW WARNING MESSAGES

13.1 POWERMETER WARNINGS

MESSAGE	DESCRIPTION
Warning: Property 'Proc_Mult_SEnergy' is being redefined, previous value overwritten <TECHLIB-359>. [PmnliNetlistInput::WarnSyntechNoLineNoFile]	Duplicate definition across Multi-VT libraries.
Warning: Found 448 instance(s) in design that can't be bound to a cell in the .lib. These instances have multiple entries for the same cell in the .lib, but the operating voltage of the instance cannot be determined. You may need to manually set the voltage of the rails connected to these instances. For these instances, the first library entry read will be chosen. See the logfile for the complete list of affected instances if there are more than three. The first few of these instances are: [PmnliNetlistInput::WarnInstsCantBind]	Expected for pad cells.
Warning: 17 cell(s) in the design have Synopsys Timing Library models but do not have internal power data. Internal power will not be calculated for these cells: [ShortDescription::WarnNoInternalData]	Expected for power pads, breaker, corner, filler, and tie cells.
Warning: 17 cell(s) in the design have Synopsys Timing Library models but do not have leakage power data. Leakage power will not be calculated for these cells: [ShortDescription::WarnNoLeakageData]	Expected for power pads, breaker, corner, filler, and tie cells.
Warning: 7 cell(s) in the design have no power data available. These cells may be missing from the libraries entirely, or may only have port views in the .cl. Instances of these cells may have switching power, but will have no internal or leakage power. The cells are: [ShortDescription::WarnNoPowerData]	Expected for fill cells.
Warning: 6 cell(s) in the design have multiple power domains and a .lib description. The tool may not be able to assign leakage power to the correct power domain. Power and ground rail reports may not match. To get around this, do not use the .lib descriptions for these cells. The cells are: [ShortDescription::WarnMultiDomainAndLib]	This message indicates cells that have multiple voltage domains but a single leakage number. PowerMeter distributes the power consumption across all voltage domains, with the result that the sum of the currents on the power rails might not match the ground rail. This can be ignored. There is no need to eliminate the .lib descriptions for these cells.

13.2 VOLTAGESTORM STATIC WARNINGS

MESSAGE	DESCRIPTION
Warning: ignore_shorts and port/label cmds are not compatible options. - ignore_shorts will be turned off.	Incompatible switches passes when running ZX within VoltageStorm. These can be ignored.
Warning: Non-manhattan geometry found.	Non-manhattan geometry warnings can be ignored.
*** Warning: Unconnected section found beginning at node 2250182, layer M1, (1441020, 1601320). *** Warning: found 1 unconnected section(s). *** Warning: Connectivity checking failed. *** Disabling current sources on unconnected sections.	VSS-only: Small disconnect in pad cell, which is common, and can be ignored. Open the **`grid_unconnected.gif`** file see the info.

13.3 VOLTAGESTORM DYNAMIC WARNINGS

MESSAGE	DESCRIPTION
Warning: ignore_shorts and port/label cmds are not compatible options. - ignore_shorts will be turned off.	Incompatible switches passes when running ZX within VoltageStorm. These can be ignored.
Warning: Non-manhattan geometry found.	Non-manhattan geometry warnings can be ignored
*** Warning: total current loaded is 0.0	This is a spurious warning, and will be removed in future releases.
*** Warning: Unconnected section found beginning at node 2254729, layer M1, (1441020, 1601320). *** Warning: found 1 unconnected section(s). *** Warning: Connectivity checking failed. *** Disabling current sources on unconnected sections.	VSS-only: Small disconnect in pad cell, which is common, and can be ignored. Open the **`grid_unconnected.gif`** file see the info.

13.4 CELTIC NDC WARNINGS

MESSAGE	DESCRIPTION
--> WARNING: Property 'Proc_Mult_SEnergy' is being redefined, previous value overwritten <TECHLIB-359>.	Duplicate definition across Multi-VT libraries.